Universidade de Aveiro
2012
Departamento de Eletrónica, Telecomunicações e Informática

Pedro Miguel dos Santos Soares

Interação entre Computadores e Placas baseadas em FPGA

Universidade de Aveiro
2012
Departamento de Eletrónica, Telecomunicações e Informática

Pedro Miguel dos Santos Soares

Interação entre Computadores e Placas baseadas em FPGA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Valeri Skliarov, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, coorientação da Professora Doutora Iouliia Skliarova, Professora Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri

presidente	Prof. Dr. Arnaldo Oliveira Professor Auxiliar da Universidade de Aveiro
orientador	Prof. Dr. Valeri Skliarov Professor Catedrático da Universidade de Aveiro
coorientadora	Prof.ª Dr.ª Iouliia Skliarova Professora Auxiliar da Universidade de Aveiro
arguente	Prof. Dr. António José Duarte Araújo Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

agradecimentos

Agradeço ao meu orientador e coorientadora, Professor Doutor Valeri Skliarov e Doutora Iouliia Skliarova por terem aceitado orientar-me e pela sua contribuição para a realização e conclusão desta dissertação.

Agradeço aos meus pais pelo amor incondicional.

Agradeço à Sara pelo seu amor, paciência e apoio.

Agradeço à minha avó por tantos almoços na sua companhia.

Agradeço à Doutora Maria do Castelo Ribeiro Biléu pela cana de pesca, que sempre irei estimar.

palavras-chave

Field-Programmable Gate Array, memória periférica, placas de prototipagem, transferência de dados, Universal Serial Bus.

resumo

Esta Dissertação é dedicada ao projeto, implementação e avaliação de ferramentas que permitem a transferência de dados entre FPGA presentes em placas de prototipagem Digilent (no caso a Nexys 2 e a Atlys) e um computador de uso geral. Esta problemática é relevante para diversos propósitos, tais como: 1) a verificação expedita de projetos para aplicações de processamento intensivo de dados; 2) para possibilitar a separação do projeto num núcleo *hardware* de elevado desempenho, que cumpra os objetivos do problema em estudo, e meios suplementares (e menos importantes) geridos em *software* executado em computadores de uso geral, tais como a geração de dados de entrada, o teste de resultados, a comparação de dados para verificação da sua correção entre outros; 3) para comparar o desempenho de qualquer tarefa executada em *software* e *hardware* através da sua execução concorrente. A Dissertação sugere duas estratégias potenciais baseadas em: 1) circuitos diretamente mapeados em hardware, e 2) microprocessadores soft core (MicroBlaze). De acordo com estas estratégias, foram propostos métodos de transferência de dados entre o hardware FPGA e software de uso geral e um número de projetos foram desenvolvidos, implementados, testados e avaliados que permitem: 1) a troca de fluxos de dados entre *software* e *hardware*, e 2) a comunicação com memórias *onboard* (DDR-RAM e flash RAM) e embutidas em FPGA. Alguns exemplos que demonstram a utilidade das técnicas propostas são igualmente apresentados.

keywords

Field-Programmable Gate Array, peripheral memory, prototyping boards, data transfer, Universal Serial Bus.

abstract

This thesis is dedicated to the design, implementation and evaluation of tools for data exchange between FPGAs of Digilent prototyping boards (namely Nexys 2 and Atlys) and a general-purpose computer. This problem is important for several purposes, such as: 1) to rapidly verify the developed projects that process data-intensive applications; 2) to split the design in a fast hardware core, solving the intended problem, and supplementary (not very important) means handled in software of general-purpose computers, such as that generate data, test the results, compare correctness of supplied data and so forth; 3) to compare performance of any particular task in software and in hardware through their concurrent execution. Generally the thesis suggests two potential strategies that are based on: 1) circuits directly mapped to hardware, and 2) soft processing (MicroBlaze) core. According to these strategies, methods of data exchange between FPGA hardware and general-purpose software were proposed and a number of projects were developed, implemented, tested and evaluated which permit: 1) to exchange streaming data between software and hardware; and 2) to communicate with onboard (DDR RAM and flash RAM) and embedded to FPGA memories. Some particular examples demonstrating usefulness of the proposed technique are also given.

Índice

Lista de Figuras

Lista de Tabelas

Lista de acrónimos

ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuits
BGA	Ball Grid Array
BMM	BlockRAM Memory Map
BSB	Base System Builder
CLB	Configurable Logic Block
CMT	Clock Management Tiles
CPLD	Complex Programmable Logic Devices
DCM	Digital Clock Manager
DDR	Double Data Rate
DSP	Digital Signal Processing
E^2PROM/EEPROM	Electrically Erasable Programmable Read-Only Memory
ECC	Error-Correcting Code
EDK	Embedded Development Kit
EPON	Ethernet Passive Optical Network
EPP	Enhanced Parallel Port
FPGA	Field-Programmable Gate Array
FSL	Fast Simplex Link
GAL	Generic-Array Logic
GPIO	General Purpose Input/Output
GPON	Gigabit-capable Passive Optical Network
GUI	Graphical User Interface
HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
HTML	HyperText Markup Language
I/O	Input/Output
IC	Integrated Circuits
IDE	Integrated Development Environment
ISE	Integrated Software Environment
JTAG	Joint Test Action Group
LAB	Logic Array Block
LC	Logic Cell
LE	Logic Element
LED	Light Emitting Diode
LMB	Local Memory Bus
LSI	Large Scale Integration
LUT	Look-Up Table
LVCMOS	Low Voltage Complementary Metal Oxide Semiconductor
LVDS	Low Voltage Differential Signaling
LVTTL	Low Voltage Transistor Transistor Logic

MAC	Multiplying Accumulator
MCB	Memory Controler Block
MMU	Memory Management Unit
MPU	MicroProcessor Unit
NRE	Non-Recurring Engineering
OEM	Original Equipment Manufacturer/Manufacturing
PAL	Programmable-Array Logic
PCB	Printed Circuit Board
PCI-e	Peripheral Component Interconnect Express
PLA	Programmable-Logic Array
PLD	Programmable Logic Devices
PLL	Phase-Locked Loop
PMOD	Peripheral Module
PRNG	Pseudo-Random Number Generator
PROM	Programmable Read-Only Memory
PS/2	Personal System/2
QFP	Quad Flat Pack
R&D	Research And Development
RAM	Random-Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RS232	Recommended Standard 232
RSDS	Reduced Swing Diferential Signal
RTL	Register Transfer Level
RTOS	Real Time Operating System
SATA	Serial Advanced Technology Attachment
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random-Access Memory
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
TTM	Time-To-Market
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
UCF	User Constraints File
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDCI	Very High-Density Cable Interconnect
VHDL	Very-high-speed integrated circuits Hardware Description Language
XPS	Xilinx Platform Studio

Capítulo 1 – Introdução

1.1 – Enquadramento

Os sistemas eletrónicos contemporâneos, fruto de décadas de evolução exponencial, estão há largos anos fortemente associados a quase todas as atividades a que o ser humano se propõe. Democratizaram-se, estando acessíveis a todos para a realização das mais diversas tarefas, tornaram-se mais ergonómicos atendendo às necessidades do utilizador final, multiplicaram as suas potencialidades ganhando recursos e eficiência a um nível que lança uma enorme sombra sobre as obsoletas referências de décadas passadas.

Entre os sistemas eletrónicos, aqueles onde a evolução é mais notória é o conjunto dos *Integrated Circuits* (IC): deixaram de ser um elemento entre vários no projeto tipo, aptos para a realização de um set reduzido de funções, para se configurarem hoje como elemento aglutinador de funcionalidades e recursos representando a base de desenvolvimento na qual assenta, hoje em dia, a esmagadora maioria da inovação tecnológica na área da eletrónica.

A maioria destes circuitos integrados que podem ser encontrados no mercado, resultado de décadas de domínio, são *Aplication-Specific Integrated Circtuits* (ASIC), ou seja IC cujas funcionalidades e recursos são por um lado orientados a uma aplicação ou objetivo e por outro nunca sofrerão alterações ou atualizações. No fundo são um exemplo de IC não programáveis. Dependendo dessa aplicação ou objetivo o ASIC poderá ser mais ou menos específico nos resultados que produz ou algoritmos que executa, mas tal não invalida que o IC em questão esteja preso às características originais de produção e aos circuitos impressos no silício e as funções que estes implementam.

Fruto do desenvolvimento na área dos IC programáveis, as *Field-Programmable Gate Array* (FPGA) são dispositivos que foram criados no sentido de permitirem reconfiguração e assim reduzirem as desvantagens que as soluções ASIC apresentam para inúmeras implementações de lógica digital. De facto por serem reconfiguráveis permitem um grau de atualização, ou seja evitam que a solução se torne rapidamente obsoleta; permitem ainda que as funcionalidades implementadas sejam profundamente alteradas, revelando-se uma excelente opção no que concerne à redução de custo.

Perante estas vantagens é compreensível o crescente domínio do mercado por soluções baseadas em FPGA e o *Researh And Development* (R&D) investido na evolução dos produtos das grandes casas produtoras, Xilinx e Altera. Ambas anunciam regularmente encontrarem-se à frente da Lei de Moore com as novas iterações das várias linhas comerciais que ambas oferecem [1][2]. Graças a processos de construção no silício cada vez mais finos as FPGA mantêm uma área reduzida, possibilitando a aplicação em inúmeras soluções embutidas, e ainda oferecem crescentes capacidades no número de portas lógicas, *Random Access Memory* (RAM) interna e ligações de alto débito com periféricos.

O utilizador final beneficia com este processo, permitindo assim um grande controlo sobre a solução que pretenda obter. Ainda mais importante, é possível alterar, refinar ou simplesmente trocar esta solução sem o constrangimento financeiro e temporal associado a qualquer um destes procedimentos numa implementação baseada em ASIC.

Perante esta realidade, em particular por possibilitar tempos muito reduzidos de projeto, de implementação e de teste o que são requisitos usuais do ambiente Universitário e de Investigação, podem-se encontrar-se cada vez mais exemplos da utilização de FPGA e de placas de prototipagem nela baseada.

Assim, soluções de telecomunicações de alto débito baseadas em fibra ótica, de transmissão de vídeo de alta definição, de controlo de voo para *Unmanned Aerial Vehicles* (UAV) ou para aplicações de radioastronomia são apenas alguns exemplos de aplicação de FPGA que beneficiam das suas características de grande paralelismo e processamento digital de grandes volumes de dados [3][4][5][6]. Em todos os casos mencionados o volume de dados e a rapidez com que estes têm que ser processados são de superior importância. Estas condições terão sido testadas previamente à colocação no mercado dessas soluções.

1.2 – Motivação

A validação e teste de soluções implementadas em FPGA depende regularmente da transmissão de elevados volumes de dados para localizações convenientes no IC ou em periféricos que lhe estejam diretamente associados, pelo que a redução do tempo de projeto e a relevância dos resultados obtidos beneficiam da facilidade e rapidez com que estas operações podem ser concluídas.

Usando placas de origem Digilent com FPGA embutidas, disponibilizadas pelo Departamento de Eletrónica, Telecomunicações e Informática, diversas Unidades Curriculares e Projetos de Investigação podem beneficiar da criação de ferramentas orientadas à geração de dados de teste aleatórios e transferência desses dados de teste mediante operações simples.

A motivação para a realização desta proposta de Dissertação está diretamente associada à premência da existência de um grupo de ferramentas que agilize o envio e receção de dados para as placas Digilent disponibilizadas, baseadas em FPGA, através da ligação *Universal Serial Bus* (USB).

1.3 – Objetivos

Pretende-se com esta Dissertação cumprir os seguintes objetivos:

- A criação de ferramentas de formatação, conversão e comparação de dados para um teste mais eficiente de resultados na área da ordenação de dados usando FPGA.
- A extensão e desenvolvimento das funções fornecidas pelo *fabric*ante Digilent, para o envio e receção de dados aleatórios com dimensão até 32 bit, através da porta USB, entre PC e placa Digilent baseada em FPGA;
- Os dados possam ser armazenados em diversas áreas de memória presentes nas placas em estudo, quer internas à FPGA, quer periféricas;

1.4 – Organização da Dissertação

Esta Dissertação encontra-se organizada da seguinte forma:

O capítulo 1 será introdutório, abordando a motivação e os objetivos propostos.

O capítulo 2 abordará as características mais relevantes das FPGA, a sua arquitetura, o fluxo de projeto e uma breve apreciação do estado da arte.

No capítulo 3 serão expostas as placas de prototipagem e as ferramentas de apoio que foram utilizadas para o estudo e desenvolvimento dos projetos em questão e também os passos necessários à sua configuração. Os diversos projetos criados e as ferramentas utilizadas são compatíveis com Windows, tendo sido testadas na versão Windows 7 de 32 e 64 bits.

O capítulo 4 visará apresentar os projetos desenvolvidos.

Segue-se o capítulo 5 que documenta os resultados associados aos projetos de transferência de dados.

O capítulo 6 conclui a Dissertação, onde serão discutidos os resultados obtidos com propostas de trabalho futuro que venham a complementar aqueles apresentados.

Nas páginas que se seguem ao capítulo 6 podem ser encontradas as referências e os anexos deste documento.

Capítulo 2 – *Field-Programmable Gate Array*

Neste capítulo pretende-se introduzir os dispositivos cuja arquitetura precede historicamente e em termos de complexidade a arquitetura das FPGA.

Em seguida apresenta um breve resumo da sua história e principais produtores, evidenciando-se os diversos elementos que compõe a arquitetura de uma FPGA moderna e aborda-se o fluxo de projeto para a implementação de *hardware* na sua estrutura lógica.

O capítulo termina mencionando o estado da arte e o futuro próximo desta área.

2.1 – Dispositivos Lógicos Programáveis

Entre os diversos tipos de IC no mercado podem definir-se duas grandes famílias: os IC ditos de lógica discreta, cuja funcionalidade é fixa no momento da sua fabricação num substrato de silício, e os dispositivos lógicos programáveis, que permitem a implementação de funções programáveis numa fase posterior. Os últimos são vantajosos porque:

- Possibilitam a implementação de projetos com uma vasta gama de envergaduras;
- Permitem não só prototipagem como também a implementação final usando o mesmo IC (a lógica discreta não possibilita tão grande liberdade, sendo necessário um grande volume de testes antes de passar à prototipagem e subsequente produção definitiva, o que implica tempo e custos na criação de novos IC para cada etapa);
- Estão aptos a uma atualização posterior comparativamente livre de limites (já uma solução de lógica discreta nunca poderá realizar operações diferentes daquelas para as quais foi produzida);
- Podem ser desenhados com apoio em lógica Booleana e linguagens descritivas de *hardware* (HDL) como o *Very-high-speed integrated circuits Hardware Description Language* (VHDL) ou o Verilog;
- Permitem tipicamente a criação de soluções de lógica combinatória, sequencial ou de memória entre outros projetos, tendo como alvo um mesmo IC para os realizar;
- Facultam usualmente a escolha de *standard* de Entrada/Saída (I/O) pretendida;
- Possibilitam a utilização de blocos de circuito já existentes, fornecidos pelo produtor ou por terceiros, para acelerar a implementação de projetos.

Apesar deste número de vantagens, os Dispositivos Lógicos Programáveis (Programmable Logic Devices - PLD) podem também ser desvantajosos em função do objetivo que se pretenda cumprir. De facto para funções muito simples podem revelar-se um investimento elevado; carecem de conhecimentos do utilizador no sentido de garantir uma configuração e desenho de implementações adequadas; finalmente, e apesar dos avanços recentes nesta área, serão ainda preteridos quando a frequência de operação requerida é muito elevada [7].

De entre os PLD podem definir-se novamente três famílias: os simples ou SPLD, os complexos ou CPLD e as FPGA se integram [8].

2.1.1 – Dispositivos Lógicos Programáveis Simples (SPLD)

Como o nome infere, são dispositivos que precedem temporalmente as versões denominadas complexas e cuja funcionalidade está presente em larga escala nestas últimas. Por representarem a génese de muitas das funcionalidades e de alguns aspetos da arquitetura presentes nas FPGA, é relevante o conhecimento de quatro tipos de SPLD para essa caracterização: os dispositivos *Programmable Read-Only Memory* (PROM), os *Programmable-Logic Array* (PLA), os *Programmable-Array Logic* (PAL) e os *Generic-Array Logic* (GAL).

O primeiro dispositivo mencionado reflete o ponto de arranque para este enquadramento. Apesar das limitações, foram historicamente dos primeiros dispositivos a permitir uma programação. Um esquema simplificado de uma PROM pode ser verificado na figura 1.

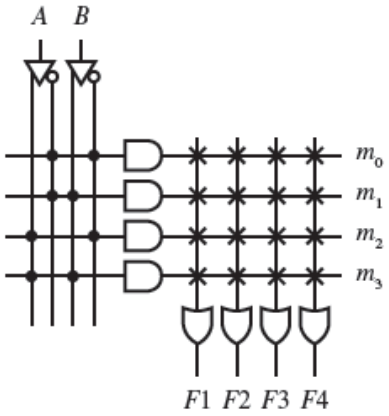

Figura 1 - Esquema simplificado de uma PROM [9]

De facto 'uma programação' reflete o sentido lato da expressão pois através de um mecanismo fusível é possível quebrar as ligações e caminhos internos condutores, programando assim o dispositivo de acordo com os requisitos do consumidor definitivamente. No sentido estrito, na medida em que para o realizar é necessário equipamento de programação adequado ao dispositivo PROM para executar a programação previamente à sua montagem no sistema final ou, alternativamente, a introdução de circuitos adequados a esta programação no sistema final.

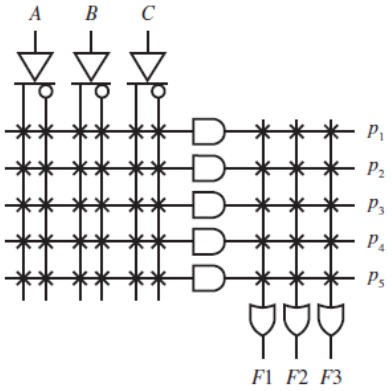

Figura 2 - Esquema simplificado de uma PLA [10]

Os casos PLA e PAL, presentes nas figura 2 e figura 3 respetivamente, consistem em IC cuja arquitetura pode ser entendida como dois planos de portas AND e OR, nos quais é possível inscrever funções booleanas compostas por estes operadores. Os portos de entrada do dispositivo estão ligados de forma conveniente a portas AND, segue-se na linha de

transmissão uma malha configurável de caminhos ligados de forma conveniente a portas OR, transmitindo o resultado das operações aos portos de saída do dispositivo.

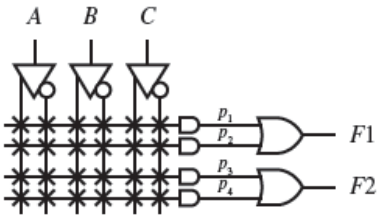

Figura 3 - Esquema simplificado de uma PAL/GAL [11]

Se no primeiro caso, PLA, tanto é possível a configuração de sinais de entrada ligados às portas AND como a arbitrariedade das operações OR realizadas sobre os primeiros resultados, no caso PAL apenas a primeira configuração é possível.

Repare-se que esta configuração traduz a possibilidade do utilizador, na forma de um utilizador final ou de uma empresa dedicada à programação em hardware sem recursos para a fabricação do IC, poder inscrever neste as funções que pretenda ficando-lhe essa configuração a partir desse momento definitivamente associada.

Estas soluções SPLD, baseadas em PROM e portanto de configuração definitiva, precedem a criação de um IC que permite este grau de liberdade, ou seja, a possibilidade der reconfiguração: os GAL, de arquitetura idêntica à PAL, mas baseados em *Electrically Erasable Programmable Read-Only Memory* (E^2PROM) o que permite a reconfiguração das operações booleanas implementadas [12].

2.1.2 – Dispositivos Lógicos Programáveis Complexos (CPLD)

Os CPLD representam uma evolução em escala quando comparados com a arquitetura SPLD: permitem a realização de um grupo de funções que necessitariam de diversos SPLD para serem realizadas usando apenas um IC, da mesma forma que um SPLD realiza as funções de uma série de pequenos IC de lógica discreta.

Assim, e no sentido de compreender o escalonamento mencionado, é correto passar a identificar-se como bloco lógico ou funcional os principais elementos constituintes das CPLD, construídos com base numa macrocélula e um circuito baseado em PAL ou PLA associado. Esta macro-célula providencia os circuitos que permitem o registo de sinais de saída e o controlo de polaridade dos sinais.

Como forma de ligação entre cada bloco lógico, e entre estes e os blocos especializados na ligação aos sinais externos ao CPLD, existe uma conexão global programável. Esta conexão programável pode ser de dois tipos: baseada numa malha de ligações cuja estrutura é idêntica à estudada para PAL e PLA, possibilitando que qualquer sinal seja conectado a qualquer bloco lógico; ou baseada num sistema de *multiplexers* ligados seletivamente às entradas das macrocélulas.

A figura 4 apresenta toda a arquitetura referida de uma forma esquemática. De seguida serão apresentadas as FPGA que como IC central ao nosso estudo merecerão uma introdução mais alargada nas secções seguintes [14].

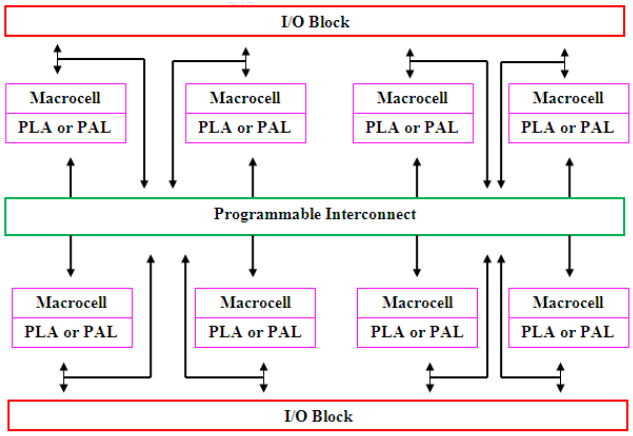

Figura 4 - Esquema simplificado da arquitetura de um CPLD [13]

2.2 – *Field-Programmable Gate Array*

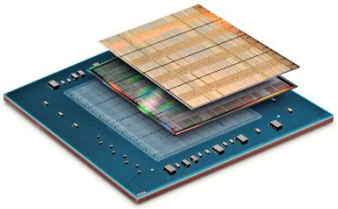

Figura 5 - Exemplo de FPGA contemporânea [15]

Nesta secção apresentar-se-á a FPGA, cobrindo um pouco da sua História, os diversos tipos, a arquitetura típica, o fluxo de projeto e finalmente o estado da arte.

2.2.1 – História

Fruto da evolução dos dispositivos mencionados nas secções anteriores, e dos desenvolvimentos introduzidos pelos fundadores daquela que é ainda hoje uma das grandes produtoras de FPGA, a primeira versão comercial deste IC foi inventada em 1985 e denominou-se XC2064. Os cofundadores chamavam-se Ross Freeman e Bernard V. Vonderschmitt e a empresa *startup* chamava-se Xilinx, não só a pioneira como ainda hoje líder de mercado FPGA, soluções *System-on-Chip* (SoC) e também de IC 3D [16].

O mercado alvo para esta empresa foi na altura o da lógica de ligação, ou seja, a produção de soluções que permitissem a ligação de circuitos *Large Scale Integrated* (LSI) entre si. Desde microprocessadores, controladores de I/O ou relógios de sistema, os diversos elementos de um sistema típico apoiavam-se usualmente numa série de IC discretos para realizarem operações que permitissem intercomunicação. Uma solução habitual era a aplicação de uma ASIC, juntando num único IC as diversas operações de ligação entre elementos do sistema, beneficiando o sistema com a fiabilidade, simplicidade e custo inferior que esta opção traz. No entanto a solução ASIC é de desenvolvimento moroso e portanto oneroso, sendo aplicados apenas em produtos construídos em volume muito elevado onde o retorno do investimento o justifique. Dado que as FPGA são IC que permitem uma fácil reprogramação, obviam a necessidade e o tempo decorrido na criação de um *layout* físico, máscara e fabricação. Desta forma apresentam vantagens no que concerne o *Time-To-Market* (TTM), ou seja em quanto tempo é possível colocar um *design* no mercado, e portanto muito reduzindo o *Non-Recurring Engineering* (NRE), o investimento inicial realizado na conceção e fabricação do IC [17].

Apostando num processo de produção *fabless*, o que significa a entrega da fabricação do IC a terceiros especializados na área ou *semiconductor foundries*, outras empresas surgiram nesta área. Merece grande relevância a maior concorrente da Xilinx, e com a qual regularmente troca a liderança de mercado em especial com o lançamento de tecnologias mais finas, a empresa Altera.

Fortemente apoiadas nas descobertas e inovações tecnológicas na área da memória, cujas motivações compreenderemos num capítulo a jusante com mais detalhe, a evolução dos produtos Xilinx ao longo dos anos levaram-nos de um dispositivo pioneiro com 64 blocos lógicos configuráveis, caracterizado por *Look-Up Table* (LUT) de 3 entradas, aos grandes IC do estado da arte com mais de um milhão de blocos lógicos configuráveis, caracterizado por LUT de 6 entradas.

Ao longo do processo evolutivo, induzidas pelo amadurecimento do paradigma de computação concorrente às FPGA que é representado pelas soluções ASIC, as empresas produtoras de FPGA foram adotando tecnologias e evoluções que lhes permitissem colmatar as lacunas existentes e aumentar as mais-valências dos seus produtos. Assim, a tecnologia de construção torna-se cada vez mais fina e foram sendo embutidos evoluídos blocos discretos, entre os quais blocos de memória, de realização de operações de processamento digital de sinal cada vez mais eficientes, de conectividade com periféricos de alto débito de dados e microcontroladores entre outras evoluções. As FPGA foram durante largos anos comparativamente simples malhas de portas lógicas configuráveis, não só compreendem hoje muito mais lógica por área de substrato semicondutor, como permitem configuração durante o seu funcionamento, efetivamente constituindo sistemas reconfiguráveis, e integram microprocessadores e periféricos no mesmo chip que outrora seriam implementados usando chips de lógica discreta.

Quando a sua arquitetura for apresentada abordar-se-ão estes blocos discretos que têm vindo a ser cada vez mais preponderantes para o desempenho e flexibilidade das FPGA, assim como se fará uma breve apresentação do estado da arte nos últimos pontos desta secção.

2.2.2 – Tipos de FPGA

Apesar de a maioria das FPGA no mercado seguir um tipo baseado em *Static Random Access Memory* (SRAM) existem outros tipos de FPGA que se baseiam noutras tecnologias para o seu funcionamento, configuração e ligações entre blocos lógicos. Apresentaremos agora as diversas alternativas.

2.2.2.1 – Tipo E²PROM

Este tipo de FPGA é constituído por células configuráveis num mecanismo similar a uma longa cadeia. Permitindo configuração quando removidas do sistema onde serão implantadas, usando dispositivos apropriados a essa programação, demoram em média o triplo do tempo neste processo que as modernas soluções SRAM.

São um tipo de FPGA de configuração não volátil, disponibilizando a configuração implementada a partir do momento em que lhes é fornecida energia. Esta situação implica que, dado que a configuração é salva no próprio IC, não são necessárias memórias periféricas para realizar qualquer recuperação da configuração. A figura 6 apresenta uma comparação simbólica entre os transístores MOS e EPROM.

As E²PROM de dois transístores e as células *Flash* possuem dimensão muito inferior ao *standard* SRAM, o que permite a construção de FPGA com área reduzida e ligações internas entre blocos lógicos de dimensão inferior, beneficiando os tempos de propagação de sinais internos da FPGA.

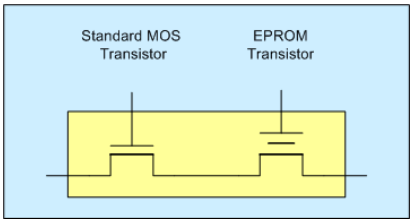

Figura 6 – Comparação entre um transístor MOS e um EPROM [18]

Por outro lado a sua construção é comparativamente mais difícil. Se as soluções estado da arte estão em linha com a Lei de Moore ou a superam, como já discutimos, as soluções E²PROM encontram-se comparativamente atrasadas em termos tecnológicos. Adicionalmente, pode ser relevante o superior consumo energético deste modelo, uma vez mais quando comparadas ao *standard* SRAM [19].

2.2.2.2 – Tipo *Antifuse*

Programável através de dispositivos específicos para o efeito, este tipo de FPGA tem em comum com o tipo E²PROM o facto de a configuração implementada não ser volátil. De facto a configuração deste tipo apoia-se num processo antagónico ao funcionamento de um fusível (como o nome indica), ou seja, cada elemento programável pode ser interpretado como um circuito aberto antes de ser programado e um circuito fechado quando programado. Esta programação consegue-se através da criação de ligações de muito baixa resistência entre planos condutores, ou seja entre os terminais, do elemento programável, como se pode observar na figura 7.

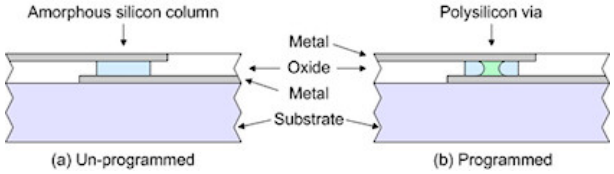

Figura 7 - Esquema de uma célula anti fusível [20]

Uma das vantagens deste tipo está associada ao facto de as células da FPGA serem mais robustas, sendo vantajosas em aplicações militares e aeroespaciais. De facto provam ser mais resistentes a radiações porque os seus elementos programáveis não comutam tão facilmente quando bombardeados por níveis de radiação anormais, situação com maior probabilidade no caso de uma célula SRAM.

Uma vez mais, graças a uma evolução comparativamente lenta, apesar de as soluções *Antifuse* oferecem características dimensionais similares às E²PROM oferecendo uma construção mais compacta, não podem por norma competir com o estado da arte SRAM.

A outra desvantagem está relacionada com elementos da FPGA que, ao contrário dos elementos programáveis, não são igualmente imunes às radiações, pelo que se recorre à implantação de sistemas de redundância por maioria, implicando elementos adicionais presentes no substrato de silício [21].

2.2.2.3 – Tipo SRAM

As células que constituem as FPGA deste género são essencialmente células de 1 bit que podem ser escritas ou lidas através de um mecanismo ilustrado pela figura 8. Uma operação de escrita de bit compreende a ativação da linha de bit e, enquanto a linha de palavra se encontrar igualmente ativa, é carregado o circuito de inversores com o valor pretendido. Graças à configuração de feedback o valor guardado é conservado. A operação de leitura compreende nova ativação da linha de palavra e o valor retido no circuito de inversores fica disponível para leitura na linha de bit.

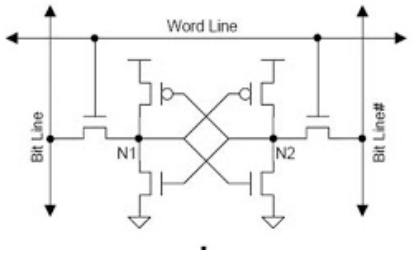

Figura 8 - Exemplo de célula SRAM [22]

Este sistema representa uma das grandes vantagens do tipo SRAM, a possibilidade de programar inúmeras vezes e do processo de programação ser rápido comparativamente às tecnologias estudadas nos pontos anteriores.

Esta tecnologia de construção tem evoluído tanto na área das FPGA como noutras áreas da computação e acaba por ditar a superior evolução que as FPGA SRAM apresentam em relação aos tipos concorrentes. Esta situação que tem um retorno positivo dado que, pelas suas características de construção e arquitetura, representam hoje uma importante referência para teste e avaliação de novos avanços tecnológicos na área da memória.

A maior desvantagem é o facto que uma vez retirada a fonte de energia ao IC as células programáveis perderem a sua configuração. Para colmatar essa eliminação é comum encontrar ligada à FPGA uma memória ou um microprocessador externo que proceda ao carregamento da programação na sua *fabric* [23].

2.2.3 – Arquitetura

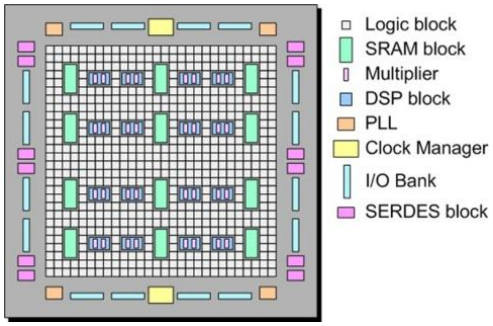

Figura 9 - A FPGA típica e os elementos da sua arquitetura [24]

A figura 9 ilustra uma disposição possível para os diversos elementos da arquitetura de uma FPGA que abordaremos nas secções seguintes. O primeiro passo é conhecer o grão da malha lógica que constitui a grande maioria dos elementos presentes numa FPGA.

2.2.3.1 – Grão

Dependendo das características dos blocos elementares programáveis que compõem a malha lógica de uma FPGA é possível caracterizá-la como sendo de grão fino, médio ou grosso. Esta caracterização permite visualizar a *fabric* e em particular os seus blocos elementares: grão relaciona blocos lógicos e a sua dimensão, ou seja, o número e a complexidade de operações que estes estão aptos a realizar. O estado da arte ao longo dos últimos anos viu as arquiteturas de grão fino, construídas por LUT e alguma lógica de suporte, dar lugar a arquiteturas de grão médio nas quais cada bloco lógico é constituído por um conjunto de LUT de 4 a 6 entradas, *multiplexers*, *flip-flops* e lógica de *carry* rápida. O estado da arte contemporâneo aponta para arquiteturas de grão médio mas o futuro pode dirigir-se no sentido do grão grosso, onde os blocos lógicos passarão a ser uma área de *fabric* típica de FPGA associada a um microprocessador. Um exemplo desta implementação pode ser encontrado na figura 10.

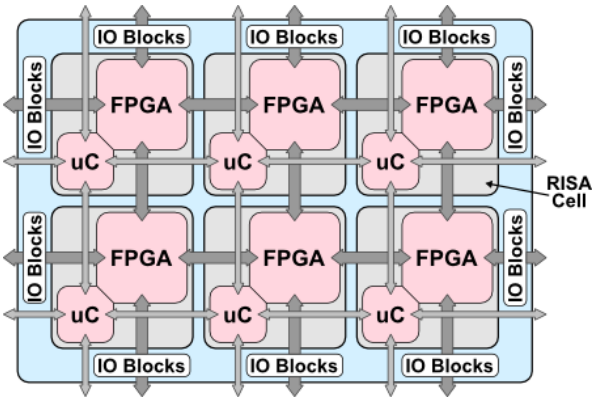

Figura 10 - Modelo de uma FPGA de grão grosso [25]

Se por um lado um grão fino dá resposta às necessidades de grande paralelismo com maior eficiência por oferecer uma malha lógica com elevado número de elementos, por outro um grão médio a grosso sofre menos penalizações no que concerne às ligações entre blocos lógicos. O grão fino e a quantidade superior de blocos lógicos impõe uma quantidade superior de interligações, aumentando os atrasos de propagação de sinais [26].

O tipo de grão em foco no estado da arte das FPGA é o grão médio, e será este o alvo do nosso estudo.

2.2.3.2 – Células Lógicas

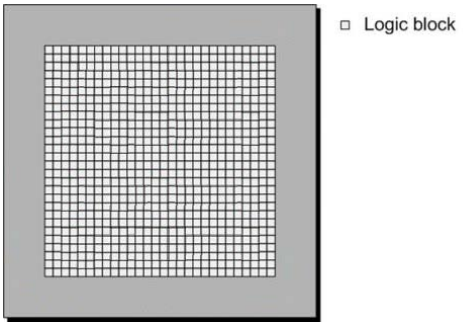

Figura 11 – Malha Lógica [27]

Os blocos lógicos que constituem as FPGA de grão médio são baseados em LUT pelo que iremos estudar com mais detalhe a sua arquitetura. A figura 11 ajuda-nos a visualizar que a malha lógica ocupa a maioria da área da FPGA.

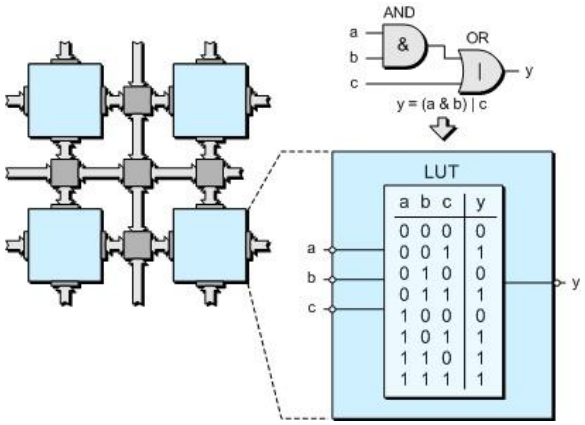

Figura 12 - Bloco lógico baseado em LUT [28]

Uma LUT é um elemento que configura a função ou funções pretendidas entre sinais de entrada no bloco lógico e sinais de saída deste. Como a sua arquitetura beneficia de uma construção realizada à custa de portas de transmissão, apresenta grande eficiência quando o volume de dados em circulação pela *fabric* é grande. A figura 12 apresenta um exemplo de implementação de uma função numa LUT de três entradas.

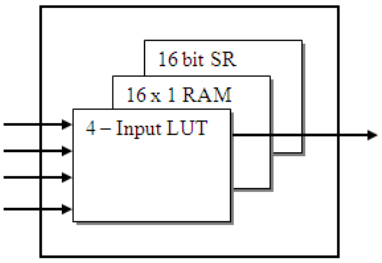

Figura 13 - As várias funcionalidades das LUT [29]

Em face da relevância deste elemento presente nos blocos lógicos de uma FPGA moderna, existem considerações adicionais que merecem destaque.

Como a sua construção é realizada utilizando células SRAM, a LUT pode para todos os efeitos ser considerada uma memória de pequena dimensão; uma LUT de quatro entradas, por exemplo, pode ser considerada uma memória de 16 bits (2^4=16). Como esta forma de memória pode ser replicada em todos os blocos lógicos ao longo da *fabric*, a *Random-Access Memory* (RAM) construída à sua custa denomina-se RAM distribuída, por oposição a outro bloco que abordaremos denominado bloco RAM onde a memória se encontra concentrada em estruturas diferenciadas.

Por outro lado, e dado que existem ligações em cadeia entre cada bloco lógico e efetivamente entre os valores que constituem a própria LUT, esta pode também ser considerada um *shift-register*; seguindo o exemplo fornecido, este *shift-register* seria de 16 bit. Em resumo, este bloco lógico exibe três comportamentos configuráveis, como observável na figura 13 [30].

Pode ainda fazer-se menção ao consenso obtido em torno das LUT com quatro a seis valores de entrada como referência nos IC contemporâneos, sendo possível encontrar no mesmo IC LUT de dimensões diferentes para responder a exigências computacionais diferentes [31].

2.2.3.3 – Hierarquia Lógica

Estes blocos lógicos que compõe a *fabric* da FPGA, foco da nossa atenção no ponto anterior, têm nomenclaturas proprietárias em função do produtor da FPGA. Se por um lado a Xilinx os identifica como *Logic Cell* (LC), a Altera considera-os *Logic Elements* (LE). No essencial ambos respeitam os conceitos já introduzidos de blocos constituídos por LUT, *multiplexers* e *flip-flops*. Uma abstração da construção destes LC ou LE e do seu enquadramento na arquitetura está patente na figura 14.

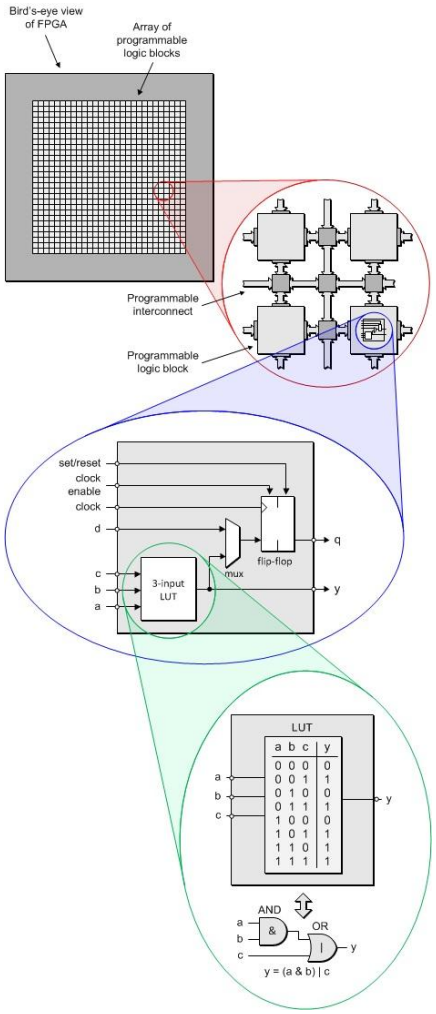

Figura 14 - A construção da malha lógica das FPGA [32]

O nível hierárquico seguinte, de acordo com a arquitetura da Xilinx, corresponde à agregação de duas LC numa *slice*. Esta associação relaciona os LC de cada *slice* através da partilha de sinais de relógio e outros sinais de controlo.

Subindo outro patamar hierárquico atinge-se o nível das *Configurable Logic Block* (CLB) ou *Logic Array Block* (LAB), caso se esteja a considerar uma FPGA XIlinx ou Altera respetivamente. Os CLB são construídos com duas a quatro *slice* e são o elemento arquitetural que serve de referência ao grão quando se pretende identificar que a FPGA é de grão médio. No fundo, e apesar das ligações programáveis que lhes são internas, os CLB ou os LAB podem ser vistos como os elementos básicos da *fabric* da FPGA, interligados por um canal programável.

Da mesma forma que é possível implementar RAM distribuída ou um *shift-register* numa LC ou LE, também um CLB constituído por diversos *slices* e cada *slice* constituído por diversos LC pode ser visto como uma RAM distribuída de maior dimensão. Seguindo o exemplo da LUT de 16 bits e propondo *slices* de duas LC e CLB de quatro *slices*, esta RAM é configurável de acordo com a tabela 1.

Tipo de RAM	Configuração
Porta Simples	16 x 8 Bit = 128 Bit
Porta Simples	32 x 4 Bit = 128 Bit
Porta Simples	64 x 2 Bit = 128 Bit
Porta Simples	128 x 1 Bit = 128 Bit
Porta Dupla	16 x 4 Bit
Porta Dupla	32 x 2 Bit
Porta Dupla	64 x 1 Bit

Tabela 1 - Configurações possíveis para a memória RAM distribuída de um CLB

Por outro lado é possível construir *shift-registers* de maior dimensão, no caso de até 128 bits.

Este aumento das capacidades entre níveis hierárquicos é conseguido graças à expansão hierárquica ser replicada nas ligações entre elementos. Entre estas destaquem-se as ligações de *carry* que permitem a realização de operações como adições, contagem ou *shift-register* de forma mais simples e eficiente, potenciando a utilização em aplicações de Processamento Digital de Sinal [33].

2.2.3.4 – Blocos embutidos: RAM

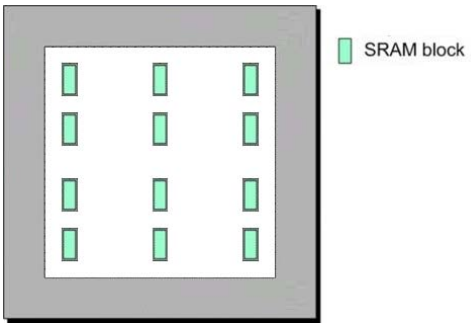

Figura 15 - Blocos RAM [34]

Devido à necessidade de RAM para a execução de inúmeros projetos, os produtores de FPGAs introduziram blocos de RAM em localizações convenientes do IC. Dependendo da configuração desejada podem ser encontrados na periferia ou espalhados pela *fabric*, de forma isolada ou em coluna, como exposto na figura 15.

Estes blocos são normalmente embutidos em maior ou menor quantidade, e de dimensão superior ou inferior, de forma proporcional à gama em que a FPGA se encontre, cobrindo usualmente necessidades entre os KByte e os MByte. Da mesma forma que é possível uma abstração que permite considerar a RAM de cada LC como uma célula de um bloco maior, compreendendo uma área da *fabric* para a realização dessa função, também os blocos de RAM embutidos permitem tal abstração, podendo ser vistos de forma individual ou como um grupo, endereçável como se de um bloco único se tratasse [35].

2.2.3.5 – Blocos Embutidos: Processamento Digital de Sinal (DSP)

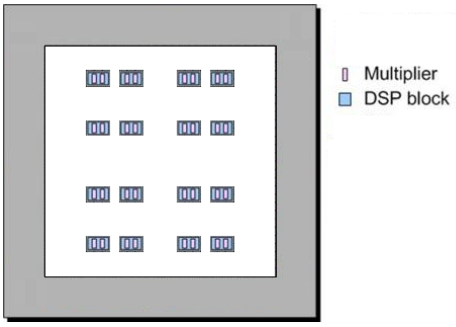

Figura 16 - Blocos DSP [36]

Apesar das funcionalidades dedicadas a agilizar aplicações de DSP (as já mencionadas ligações de *carry* tanto inter como intra grão) a realização de multiplicações representa um desafio para a *fabric* da FPGA. Para obviar esta situação foram desenvolvidos blocos DSP especificamente desenhados para endereçar estas operações aritméticas e outras com similar peso para a *fabric*. Além das vantagens em performance obtidas pela realização de uma operação de multiplicação num bloco discreto, por oposição à *fabric* da FPGA, é por norma a área de *fabric* que seria ocupada para a realização dessa operação, e que desta forma fica disponível para a programação de um projeto de maior dimensão, que representa o maior benefício. Uma distribuição possível destes blocos está considerada na figura 16.

Para conseguir a deslocalização deste tipo de operações da *fabric* os blocos DSP são construídos para a realização de adições, multiplicações e operações *Multiply And Accumulate* (MAC) de vetores de dimensão média a grande, e posicionados estrategicamente perto dos blocos RAM embutidos referidos no ponto anterior. Esta associação espacial acontece dado que a operação dos primeiros requer regularmente a operação do segundos e a proximidade beneficia os atrasos de propagação de sinais entre ambos blocos [37].

2.2.3.6 – Blocos embutidos: microprocessador

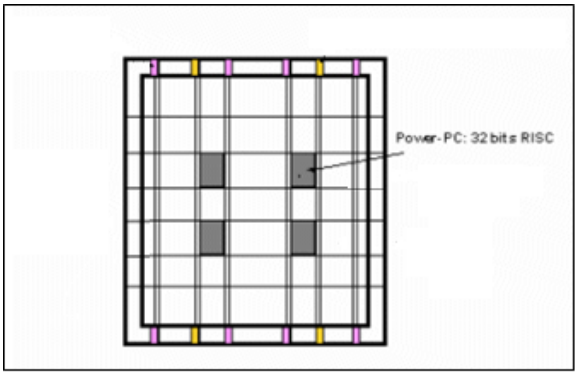

Figura 17 - Blocos microprocessador [38]

Para permitir um aumento de complexidade e de controlo das operações realizadas pela FPGA, em particular para potenciar a execução de rotinas de elevada complexidade ou desenhadas em código em linguagens de alto nível não descritivas de *hardware*, a FPGA está regularmente associado a um IC exímio nessas tarefas: o microprocessador.

Usualmente embutido na *Printed Circuit Board* (PCB) próximo da FPGA, para servir as operações que ocorram na sua *fabric* ou servir de controlador dos canais de comunicação entre FPGA e periféricos, os microprocessadores estão expostos a penalizações ao seu desempenho quer pela distância quer pela profusão de ligações passíveis de falha ou interferência até à FPGA.

Este problema foi atacado deslocando os microprocessadores para o interior da FPGA, embutindo-os da mesma forma que os blocos embutidos discutidos nos pontos anteriores ou instanciando-os usando as capacidades lógicas da *fabric* para recriar o seu funcionamento. Graças aos avanços na construção fina de circuitos integrados, as gamas altas das grandes produtoras de FPGA incluem estes microprocessadores como blocos discretos, denominados *Hard Core*. As soluções implementadas na *fabric*, mais comuns em gamas baixas (volume) a médias (*consumer*), são denominadas *Soft*/*Firm Core*.

Pode-se neste ponto verificar novamente a extrema flexibilidade de uma solução baseada em FPGA por oposição a uma solução baseada em IC de lógica discreta. Um processador embutido na versão *Soft* ou *Firm Core* pode ser instanciado por um período de tempo no sentido de controlar uma série de periféricos essenciais a determinada aplicação, sendo posteriormente possível atualizar o *software* de controlo desses mesmos periféricos ou simplesmente endereçar um diferente grupo destes últimos para outra aplicação díspar mediante uma simples reprogramação da FPGA.

Tornam-se ainda ultrapassadas em alguma medida as questões da compatibilidade e *trade-offs* associados às soluções ASIC, que privilegia em muitos casos um determinado segmento de mercado e um determinado conjunto de características que as tornam aptas às tarefas típicas desse segmento. No caso de um processador embutido em FPGA uma alteração das características da configuração (*software*) poderão permitir resultados idênticos aos da troca de ASIC microprocessador e controladores de periféricos com um TTM e um NRE substancialmente reduzidos [39].

Dado que parte do nosso estudo se foca na utilização de um destes blocos microprocessador, em particular de uma solução *Soft Core*, o seu estudo será mais aprofundado nos pontos que se seguem.

2.2.3.6.1 – Blocos embutidos microprocessador: *Hard Core*

Uma solução *Hard Core* é uma implementação física, ou seja um ou vários blocos dedicados que podem popular tanto a periferia da *fabric* como o seu centro ou outras áreas convenientes.

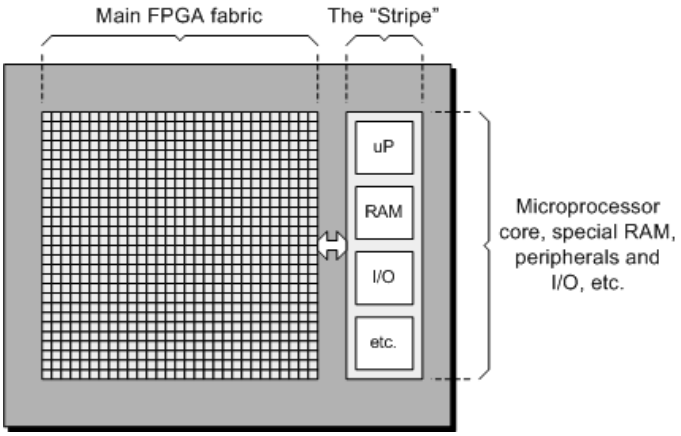

Figura 18 - *Stripe* [40]

Em qualquer caso, e dado que ambos os IC (microprocessador e FPGA) passam a coexistir no mesmo substrato ou pelo menos no mesmo encapsulamento, o produto final resulta em soluções ou sistemas cujos PCB terão uma área mais reduzida e um custo inferior, um consumo energético inferior comparativamente a soluções com os dois IC embutidos de forma discreta num PCB, uma complexidade de ligação entre elementos inferior e portanto fiabilidade e performance superior.

A primeira alternativa, colocando um bloco na periferia, beneficia a contenção de custos na produção das FPGA já que a *fabric* de várias linhas de produtos podem ser construídas de forma igual, variando apenas a área da FPGA onde os blocos adicionais serão embutidos, denominada *Stripe*. Esta opção está esquematizada na figura 18.

Apresenta ainda outra vantagem que reside na possibilidade de na área adicional destinada a albergar o bloco Hard Core poderem ser também introduzidos blocos adicionais de memória dedicada ou blocos de conectividade com periféricos com funcionalidades especiais entre outros.

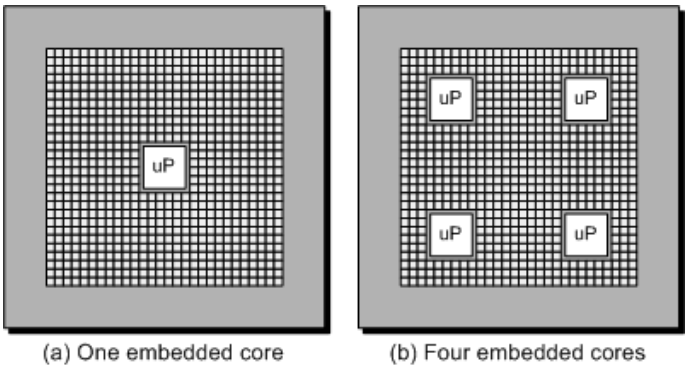

Figura 19 - Posicionamento dos blocos microprocessador embutidos na *fabric* [41]

A segunda alternativa, colocando um ou mais blocos em locus adequados no interior da *fabric* da FPGA, é vantajosa por colocar o(s) Hard Core tão próximo(s) quanto possível da *fabric*, potencialmente reduzindo os tempos de atraso nas comunicações internas deste IC. Esta implementação implica por norma que blocos adicionais que uma *Stripe* permite embutir não sejam aplicados, sendo as funcionalidades de memória e de ligações externas

realizadas com blocos discretos já presentes algures ao longo da *fabric* [42]. A figura 19 ilustra esta disposição.

2.2.3.6.2 – Blocos embutidos microprocessador: *Soft/Firm Core*

FPGA sem blocos microprocessador embutidos podem beneficiar do funcionamento de um microprocessador usando uma versão configurada na própria *fabric* sendo estas soluções designadas por *Soft Core* ou *Firm Core*. Para todos os efeitos, são implementações em áreas da FPGA não diferenciadas nem exclusivas para a realização deste fim.

Comparativamente mais lentos que a alternativa dedicada (em particular em FPGA de gamas baixas operando a frequências inferiores) apresentam no entanto maior flexibilidade. Até à exaustão de recursos é possível instanciar tantos núcleos quantos sejam necessários, a sua atualização é bastante acessível bem como é possível a troca de periféricos alvo mediante a troca de *software* que implementa esses controladores, entre outras vantagens.

A diferença entre uma solução *Soft Core* e uma *Firm Core* prende-se com o tipo de implementação: se o *Core* é fornecido como uma *netlist Register-Transfer Level* (RTL) que será sintetizada com a restante lógica pretendida trata-se um *Soft Core*; caso o núcleo seja fornecido como um conjunto de blocos lógicos cuja área já se encontra pré-mapeada e as ligações pré-encaminhadas na *fabric*, denomina-se *Firm Core*. A solução com maior ênfase nas FPGA alvo desta Dissertação é *Soft Core*, designa-se Xilinx MicroBlaze e merecerá destaque especial [43].

2.2.3.6.3 – Microprocessador *Soft Core*: Xilinx MicroBlaze

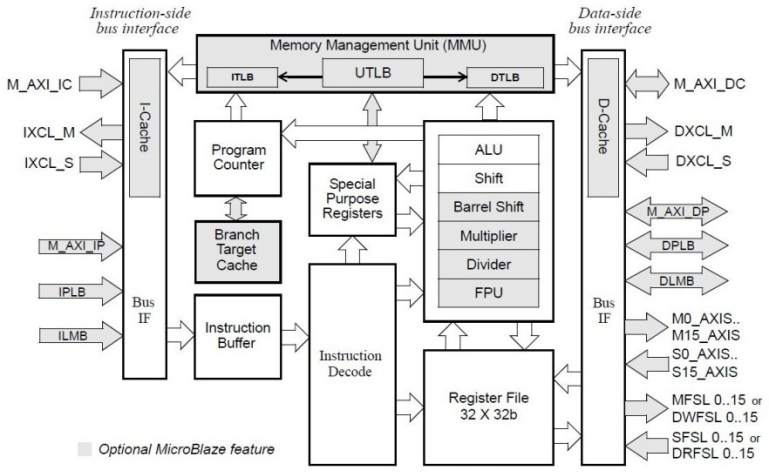

Figura 20 - Arquitetura do MicroProcessador Soft Core MicroBlaze [44]

Microblaze é a designação de um processador *Soft Core* RISC (Reduced Intruction Set Computer) 32 bit de arquitetura Harvard implementável na *fabric* de FPGAs Xilinx, para as quais está otimizado. Esta arquitetura implica que existem dois canais 32 bit, um para dados

e outro para instruções. A figura 20 apresenta um diagrama de blocos funcional do núcleo MicroBlaze, entre blocos indispensáveis e blocos opcionais, refletindo as configurações possíveis e mostrando que é possível o desenho de um microprocessador à medida das necessidades do projeto.

A arquitetura que suporta esta flexibilidade inclui um *pipeline* de 3 etapas, 32 registos de uso geral, uma *Arithmetic Logic Unit* (ALU), uma unidade de deslocamento e dois níveis de atendimento a interrupções entre outras características. Dada a profusão de recursos disponibilizados, serão apresentados os mais relevantes na lista que se segue [45]:

- Modo de interrupção de baixa latência;
- *Fault Tolerance*, mecanismo de deteção de erros em blocos RAM, incluindo *Error Correction Codes* (ECC) e suporte *Lockstep* (*debug* através da execução de mais do que um core MicroBlaze);
- Memória do tipo bloco RAM *Local Memory Bus* (LMB);
- Proteção por paridade em blocos RAM e *caches*;
- Compatibilidade com a norma IEEE-754 (aritmética de vírgula flutuante);
- Precisão simples nas operações de aritmética de vírgula flutuante;
- Unidade de Gestão de Memória (MMU);
- MMU com memória virtual suportadas pelo sistema operativo Linux;
- Modo *MicroProcessor Unit* (MPU) para execução de zonas críticas de código em aplicações *Real Time Operating System* (RTOS);
- *Cache* para dados e instruções de dimensão configurável entre blocos RAM e RAM distribuída;
- Operações de *write-through* e *write-back* mapeadas diretamente (*cache*);
- *Victim Cache* configurável de duas a oito linhas;
- Otimizações para operações condicionais;
- Lógica de predição de operações condicionais;
- *Cache* de alvos de operações condicionais;
- Aceleracção de *hardware* de execução;
- *Barrel-shifter* de operação em um ciclo;
- Divisor de inteiros de operação em 32 ciclos;
- Multiplicador de operação em um ciclo;
- Instruções de comparação de padrões;
- Registos *Machine Status Set* e *Clear*;
- Acesso atómico;
- Suporte à conversão do tipo de *endian* das operações;
- Suporte à identificação de exceções de *hardware*;
- Captura de acessos à área de endereçamento desalinhados;
- Captura de instruções ilegais;
- Captura de erros do canal de dados;
- Captura de erros do canal de instruções;
- Captura de exceções de divisão;
- Captura de exceções de vírgula flutuante;
- Captura de exceções de FSL;
- Captura de exceções de MMU;

- Sinalização de interrupções do tipo nível ou transição;
- Lógica de *debug*;
- Controlo JTAG através de um núcleo de suporte ao *debug*;
- Até oito pontos de paragem de *hardware*.

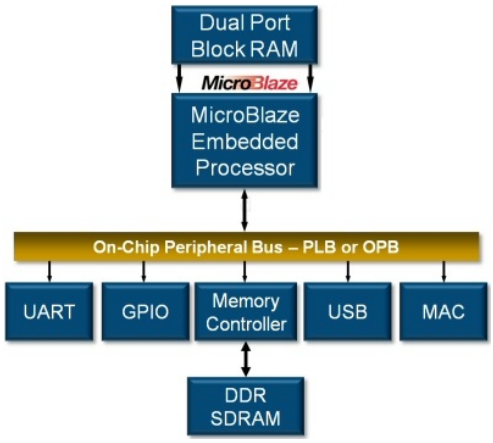

Figura 21 - Ligação entre microprocessador e periféricos [46]

A figura 21 apresenta a arquitetura das ligações entre o microprocessador MicroBlaze e os periféricos: é realizada através de um interface de periféricos (especificação PLB, OPB ou AXI), disponibilizando controladores específicos para o atendimento a periféricos de características exigentes, como memórias de alto débito [47].

Merece ainda referência o protocolo de canal de comunicação ponto-a-ponto unidirecional *Fast Simplex Link* (FSL) destinado a estabelecer ligações de débito elevado entre o microprocessador instanciado e o restante *hardware* a operar na FPGA [48].

2.2.3.7 – Árvores de relógio e blocos embutidos *Digital Clock Manager*

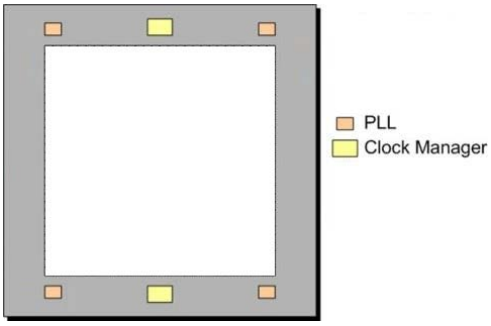

Figura 22 - *Phase-Locked Loops* e *Clock Managers* [49]

Por forma a garantir a disponibilização de sinais de relógio a todos os elementos que o requerem para a sua operação, podem encontrar-se estruturas dedicadas à sua aquisição de

fontes externas e ao encaminhamento ao longo da FPGA até cada elemento. A figura 22 apresenta a disposição possível dos blocos que controlam essas funcionalidades.

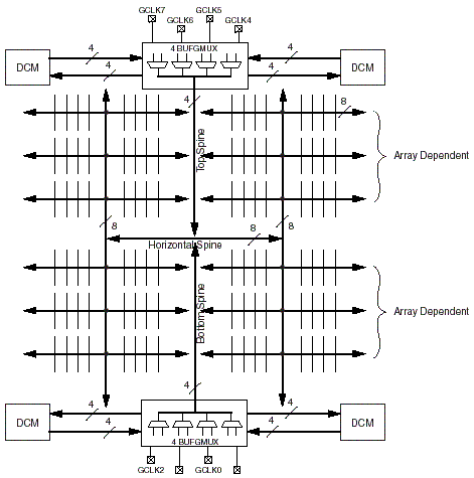

Figura 23 - Árvore de relógio [50]

Para realizar este exercício são integradas árvores de relógio com uma distribuição pela *fabric* que abre em subsequentes ramos do ponto de entrada da FPGA até às entradas de relógio individuais de cada elemento, numa estrutura desenhada para que os diversos caminhos percorridos pelo sinal de relógio sejam de distância muito similar. Um exemplo gráfico está disponível na figura 23. A motivação para este cuidado no desenho das diversas árvores presentes numa FPGA é garantir que os sinais de relógio servidos aos vários pontos do IC sejam tão síncronos quanto possível.

As árvores de relógio representam apenas a primeira linha de ação adotada no desenho e construção de FPGA no que concerne os sinais de relógio que a podem percorrer. Dada a relevância dos sinais de relógio para a esmagadora dos projetos implementáveis na *fabric* existe um ou mais blocos discretos de gestão de relógio ou *Digital Clock Manager* (DCM).

Estes não só adquirem o sinal de relógio externo como realizam uma série de outras operações destinadas a corrigir o sinal original e flexibilizar os relógios internos disponibilizados para implementação ao utilizador final. O plural neste caso implica que estas unidades são capazes de gerar, com base no relógio original, vários novos sinais de relógio para uso tanto interno (sendo aplicados por exemplo num grupo de CLB ou LAB) como externo (sendo fornecido por exemplo a uma ADC como relógio referência). Esta unidade apresenta regularmente as seguintes funções: redução do *jitter* do sinal de entrada; síntese configurável de frequências, a partir da frequência original; mudança configurável de fase em relação à fase do relógio original; correção do atraso de propagação dos relógios secundários em relação ao original, introduzido no processamento da própria DCM e dos caminhos percorridos [51].

2.2.3.8 – Blocos embutidos: *General Purpose Input/Output*

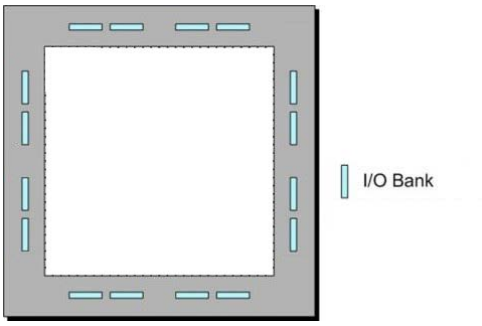

Figura 24 - Blocos GPIO [52]

No sentido de potenciar a compatibilidade entre FPGA e a avassaladora quantidade de IC no mercado, assim como a acessibilidade da lógica a sinais externos, e portanto a uma grande quantidade de protocolos de comunicação distintos, as ligações entre FPGA e os circuitos que lhe são periféricos são estabelecidas através de blocos dedicados na periferia da *fabric*. Os pinos ou *pads* desses blocos são configuráveis em função desses mesmos protocolos, estando distribuídos por bancos. Uma disposição possível desses bancos na periferia da *fabric* é ilustrada na figura 24.

Dadas as características de alguns protocolos contemporâneos, como o alto débito a que funcionam de forma unipolar ou diferencial, ou a reduzida distância entre pinos ou *pads* dos IC que usam esses protocolos, estes bancos de pinos de I/O possibilitam a introdução de resistências de terminação internas de valor nominal configurável adaptáveis aos protocolos suportados, para evitar reflexões que degradassem os sinais [53].

2.2.3.9 – Blocos embutidos: *Gigabit Transceivers*

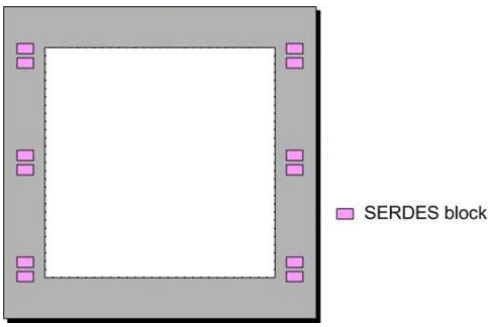

Figura 25 - Blocos *GigaBit Transceivers* [54]

Estes blocos são responsáveis por conectividade série de alto débito, usando canais diferenciais de envio e receção, especializados no envio e receção de grandes volumes de dados, sendo atualmente comuns valores de largura de banda na ordem da dezena de Gb/s. A sua colocação pode ser realizada em pontos da periferia como os ilustrados na figura 25. Esta conectividade é assim realizada de forma mais imune ao ruído e sem a constrição que estaria

associada à utilização de uma grande quantidade de pinos que seriam necessários para a criação de canais de comunicações de desempenho idêntico [55].

2.2.4 – *Intellectual Property Cores* (IP *Cores*)

Para reduzir o tempo de desenvolvimento e projetar soluções fiáveis existem no mercado Núcleos de Propriedade Intelectual que consistem, para a realidade dos projetos para FPGA, em blocos lógicos implementáveis em *hardware* desenhados, testados e validados pelo criador ou por terceiros para a realização de uma ou várias funções.

As grandes produtoras de FPGA já mencionadas, Xilinx e Altera, disponibilizam com os seus pacotes de *software* CAD inúmeros IP *Cores* que implementam desde memórias em blocos RAM até protocolos de comunicação complexos como PCI-Express ou soluções de *Forward Error Correction*. A existência e disponibilização destes núcleos permite a aceleração de novos projetos através da reutilização.

O microprocessador *Soft Core* MicroBlaze pode ser considerado um IP *Core*, disponibilizado pela Xilinx para as suas FPGA [56].

2.2.5 – Fluxo de projeto

O projeto de implementações FPGA passa por diversos passos, alguns comuns ao processo de implementação num ASIC. Existem diferenças importantes no processo de criação de um ASIC e de uma FPGA que muito beneficiam o seu TTM e NRE, serão apresentados ambos os fluxos para uma apreciação comparativa. Sendo cronologicamente precedente ao FPGA, será apresentado o processo para os ASIC.

Como apresentado na figura 26 podem resumir-se os passos deste fluxo da seguinte forma [58]:

- *Design Entry*: a entrada do projeto na aplicação adequada à sua síntese, criada numa linguagem adequada à descrição do *hardware* (p.e. VHDL ou Verilog) ou através de esquemáticos descritivos;
- *Logic Synthesis*: a síntese da *netlist* extraída do projeto realizado no ponto anterior, descrevendo as células lógicas, interligações e comportamento que o design infere;
- *System Partitioning*: divide um sistema complexo nas células e blocos lógicos do tipo ASIC disponíveis para implementação;
- *Prelayout (Behavioral) Simulation*: simulação destinada a validar o funcionamento virtual do circuito;
- *Floorplanning*: colocação de blocos de lógica em áreas físicas convenientes do substrato do IC;
- *Placement*: especificação da localização adequada das células que constituem os blocos de lógica;
- *Routing*: criação das interligações;

- *Extraction (Back Annotation)*: determinação de características de impedância e capacitância das ligações criadas, no sentido de estimar os atrasos dos sinais entre blocos, para efeitos de simulação;
- *Postlayout (Physical) Simulation*: verificação física do funcionamento do circuito, usando já as informações relativas aos atrasos de propagação calculados no ponto anterior;
- *Design Rule Check (DRC)*: verificação do *layout*, contrastando-o com as especificações pretendidas o design.

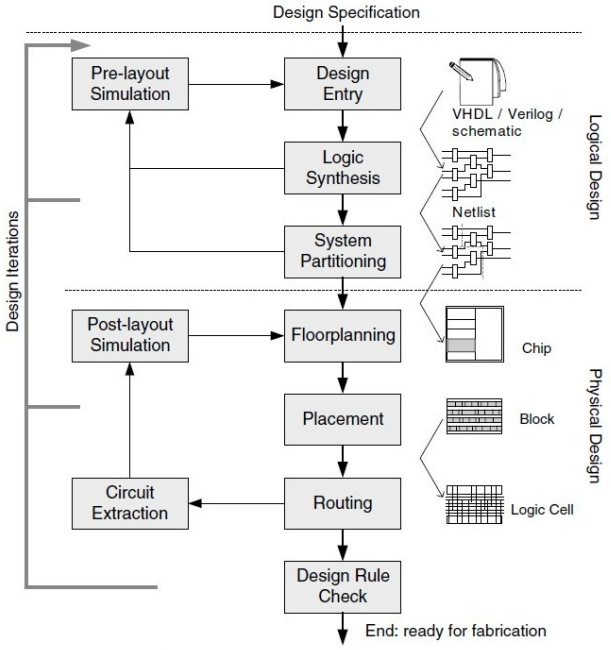

Figura 26 - Fluxo de Projeto - caso ASIC [57]

Existe uma série de pontos deste processo que são designados iterações do projeto, como exposto na figura 26. De facto estes pontos representam alguns dos momentos do projeto onde será provável a verificação de desvios face aos requisitos do sistema, forçando o recomeço parcial ou total do processo de projeto. Como parte do fluxo de projeto implica a fabricação de um IC físico não reprogramável, e apesar da existência de um grande número de blocos comerciais implementáveis testados e validados, será bastante considerável a duração deste fluxo em particular na fase de prototipagem física do IC.

Contraste-se o fluxo de projeto ASIC com o fluxo de projeto FPGA apresentado na figura 27 [60]:

- Realiza-se igualmente uma entrada do projeto através de um suporte adequado;
- Segue-se uma síntese desse projeto, convertendo o suporte utilizado para primitivas de baixo nível adaptadas à FPGA alvo;
- Mapeiam-se os elementos lógicos em elementos da arquitetura da FPGA efetivamente capazes de realizar a função desenhada;
- Posiciona-se nos componentes físicos da FPGA alvo os elementos mapeados;

- Realiza-se o encaminhamento das ligações entre elementos;
- Gera-se um ficheiro de programação da FPGA, designado *BitStream*.
- Programa-se a FPGA, carregando o *BitStream* criado;

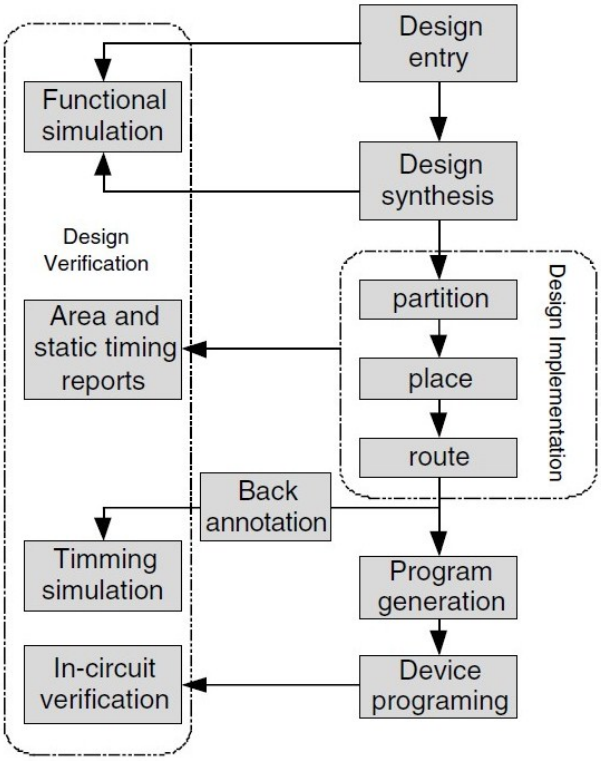

Figura 27 - Fluxo de Projeto - caso FPGA [59]

Ao longo deste procedimento é possível a realização de verificações do *design* em vários momentos, numa possibilidade similar ao caso ASIC. Estas podem tanto ser do tipo funcional numa fase introdutória, como da ocupação da área lógica do FPGA, da estimativa e simulação dos tempos e atrasos de operação até à verificação no próprio circuito, mediante a programação de lógica para a realização dessa tarefa adicional.

Se à primeira vista ambos os diagramas de fluxo são similares, excluindo os passos de programação do dispositivo e da verificação interna ao circuito o fluxo de projeto para uma FPGA é passível de execução exclusiva num ambiente CAD adequado. Caso a implementação não seja adequada aos requisitos do *design* uma reprogramação de uma FPGA é muitas ordens de grandeza mais rápida que a prototipagem de novo ASIC.

A figura 28 apresenta igualmente um fluxo de projeto mas neste caso para implementações MicroBlaze. Note-se na área esquerda da figura a entrada de bibliotecas do utilizador e de ficheiros de aplicação de *software*, no fundo o código que será compilado no elemento *Software Design*. Na área à direita está patente a entrada de um repositório de IP *Cores* referentes aos diversos elementos de *hardware* a implementar, sendo da responsabilidade do elemento de *Hardware Design* a criação das *netlists* e sínteses indispensáveis a esse fim.

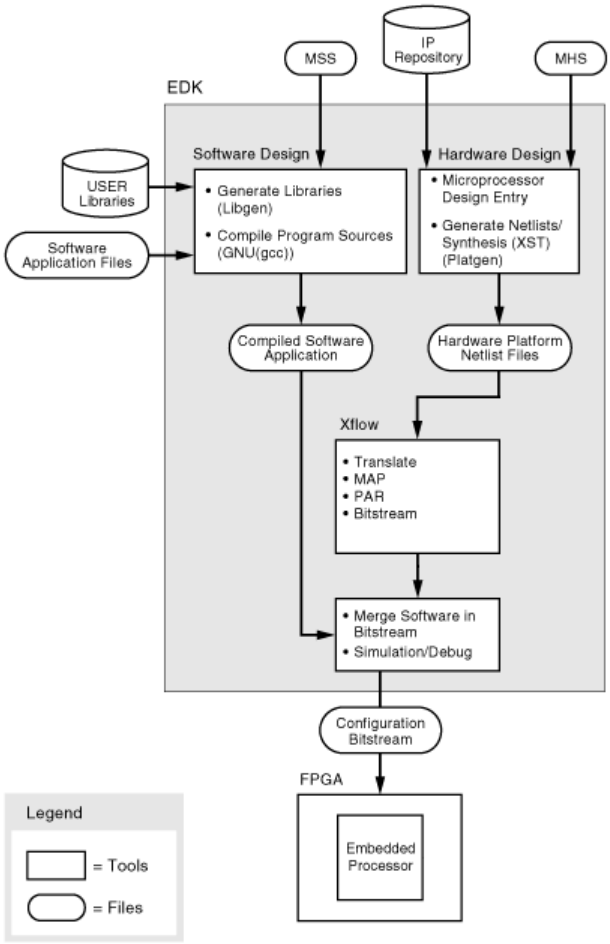

Figura 28 - Fluxo de Projeto - Implementações MicroBlaze [61]

O elemento *Compiled Software Application* representa o ficheiro ELF, com os algoritmos e funções que serão executados pela implementação MicroBlaze. Este ficheiro e o ficheiro *BitStream*, que contém a informação acerca do hardware a implementar, dão origem assim à programação da FPGA.

2.2.6 – Estado da arte

Mantendo a referência nas grandes empresas líderes de mercado, Xilinx e Altera, duas gamas surgem como exemplos do estado da arte: Virtex-7 e Stratix-V respetivamente.

Dotadas de tecnologia de construção de 28 nm apresentam características impressionantes comparativamente às gerações a que sucedem mas também dentro da família dos IC.

Com mais de 1.000.000 de células lógicas, as Virtex-7 e Stratix-V que se encontram no topo da gama são recordistas na dimensão da *fabric* mas também no que concerne a todos os blocos discretos que incorporam: mais de 50 Mbits de memória interna em blocos RAM, núcleos dedicados *Peripheral Component Interconnect Express* (PCIe) de terceira geração, interfaces para DDR3-SDRAM operando a frequências até 1866 MHz, centenas de blocos

multiplicadores de precisão variável que permitem até 5335 GMAC/s, dezenas de *transceivers* a operar até 28,05 Gb/s, mais de um milhar de pinos de I/O e dezenas de blocos de gestão de relógio constituem IC de grande desempenho. Apesar destas características e recursos foi possível reduzir o consumo energético global das novas gerações [61][62].

O avanço tecnológico da fabricação permite o desenvolvimento deste tipo de soluções e ainda o encaminhamento para soluções *System-on-Chip* (SoC), onde a FPGA não é o elemento central do sistema complexo. De facto existem e estão a ser estudadas mais soluções que incluem num mesmo *chip* a *fabric* já estudada, blocos típicos comuns a FPGA e ASIC (blocos de memória, blocos DSP entre outros) e microprocessadores avançados, reunindo desta forma o melhor de dois mundos: a performance ASIC e a flexibilidade FPGA. Um exemplo é o dispositivo Xilinx Zynq-7000, cuja arquitetura está ilustrada na figura 29. Apesar da presença de lógica programável o foco deste IC está colocado no microprocessador dedicado Cortex-A9, de elevado desempenho, sendo a lógica programável mencionada destinada a aumentar a flexibilidade e os recursos desta plataforma [64].

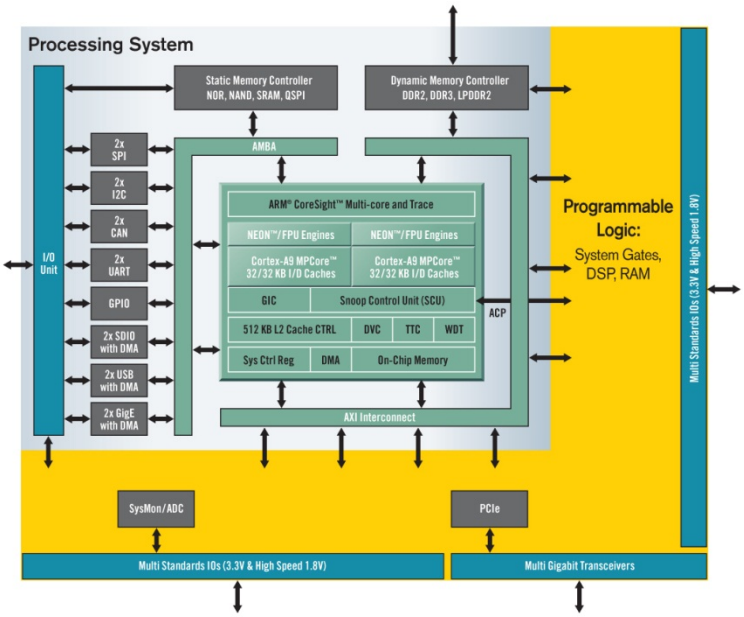

Figura 29 - SoC Zynq-7000 [65]

2.3 – Conclusão

Neste capítulo foram apresentadas as FPGA:

- Conhecimento da sua história e enquadramento nos PLD;
- Os vários tipos de FPGA no mercado;
- A arquitetura típica das FPGA contemporâneas e os diversos blocos que as constituem;
- O fluxo de projeto típico para programação deste IC;
- Alguns elementos do estado da arte.

No próximo capítulo iremos conhecer as ferramentas utilizadas para a criação dos vários projetos que serão apresentados no capítulo 4.

Capítulo 3 – Sistemas de desenvolvimento

Neste capítulo abordaremos os sistemas de desenvolvimento utilizados para a realização dos projetos a que nos propusemos no primeiro capítulo.

Para isso cobriremos primeiro o *hardware* alvo, no caso produtos da Digilent disponibilizados pelo Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, produtos esses adquiridos em quantidade para efeitos de utilização em Unidades Curriculares e Projetos de Investigação do Departamento. Serão abordadas não só as FPGA que estão no centro físico e lógico de cada uma das duas placas utilizadas como também os periféricos que a estas se encontram ligados e associados.

Em seguida faz-se uma exposição do *software* utilizado para a programação da FPGA e de aplicações de consola que realizam operações no PC que serve de anfitrião à utilização das placas. Em cada caso serão abordadas as configurações e procedimentos usados para cada aplicação.

3.1 – Placas de prototipagem Digilent

A Digilent Incorporated é uma empresa Norte-Americana, fundada por dois Professores do Ensino Superior desse país, orientada a preencher a necessidade de soluções que aumentassem a produtividade e sucesso das Unidades Curriculares sob a sua alçada. O *Mission Statement* desta empresa expressa a intensão de disponibilizar, a Professores e Alunos, ferramentas de baixo custo para desenho de projetos de Engenharia Eletrónica. Fundada no ano 2000, esta empresa orgulha-se de estar presente em mais de 2000 Universidades espalhadas por 70 países o que reflete o seu cliente alvo tal como o seu sucesso no mercado. Sucesso patente ainda nas soluções de desenho e produção *Original Equipment Manufacturer* (OEM) fornecidas a empresas como a Xilinx, a Analog Devices ou a Cypress Semiconductor.

Considerando o ambiente alvo, as placas produzidas pela Digilent para distribuição em nome próprio ou por terceiros primam pelos preços competitivos, pela possibilidade de expansão das características e funcionalidades originais dos produtos e pela disponibilização de uma gama de ferramentas de apoio à sua utilização.

O primeiro aspeto verte-se num equilíbrio entre capacidades e custo do IC aplicado nas placas produzidas por esta empresa, que seleciona por norma elementos de gama média-baixa a gama média-alta como forma de manter os preços de venda ao público acessíveis, em particular através de um programa de descontos Académicos, sem prescindir de uma gama de produtos destinada a requisitos de desempenho diferentes.

O aspeto seguinte considera as capacidades de expansão disponibilizadas para cada placa que, utilizando ligações populares e de fácil adoção para protocolos de reduzido débito e portas de maior qualidade para débitos elevados, permitem a ligação de uma série de *daughterboards* não só da própria Digilent Inc. como de outras empresas.

Finalmente é disponibilizado um conjunto de ferramentas que compreendem aplicações para um fácil carregamento da programação nos IC, *Software Development Kit* (SDK) e documentação associada que permitem explorar as capacidades das placas [66].

As placas sobre as quais nos debruçámos foram a Nexys 2 - 1200 e a Atlys, placas com um valor de mercado de 189 e 349 dólares respetivamente (ou 139 e 199 dólares na eventualidade de aquisição mediante o desconto Académico) e que de seguida passamos a apresentar [67][68].

3.1.1 – Digilent Nexys 2 - 1200

A Nexys 2 - 1200 é uma placa baseada num FPGA Xilinx, neste caso um modelo Spartan-3E, IC de gama baixa lançada no ano de 2005. A figura 30 demonstra o aspeto e a organização da placa em questão: ao centro encontra-se a FPGA, à sua volta em locus vantajosos encontram-se os controladores e periféricos que lhe estão ligados e na periferia as diversas ligações externas através de conectores apropriados.

De seguida apresenta-se um breve resumo das características mais relevantes dos componentes presentes na Nexys 2.

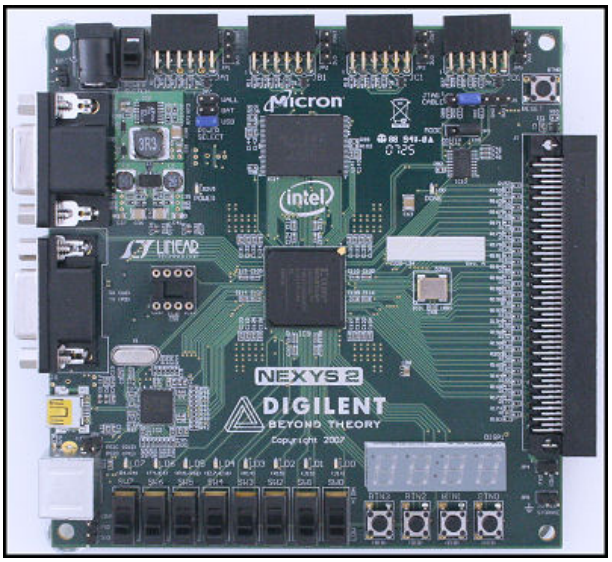

Figura 30 - Placa de prototipagem Digilent Nexys 2 [69]

3.1.1.1 – FPGA Xilinx XC3S1200E (Spartan-3E 1200)

A família de FPGA Spartan-3E é constituída por seis elementos que cobrem necessidades de projeto entre as 100 mil e as 1,6 milhões de portas lógicas, e representa uma evolução das características de largura de banda e funcionalidade da família Spartan-3, apresentando mais lógica por I/O. O mercado alvo desta família inclui eletrónica orientada ao acesso à internet por banda-larga, redes domésticas ou equipamento de televisão digital, no fundo refletindo aplicações de baixo custo orientadas ao consumidor e de relativamente reduzida complexidade lógica e de processamento.

Esta família de IC apresenta as seguintes características mais relevantes:

- Realizada em tecnologia 90 nm;
- Suporta diversos protocolos de comunicação através de até 376 pinos de I/O disponibilizados para sinais unipolares como por exemplo LVCMOS ou LVTTL;
- Até 156 sinais diferenciais para sinais do tipo LVDS ou RSDS;
- A sinalização destes sinais pode ser configurada entre 3,3 Volts e 1,2 Volts, dependendo do protocolo adotado;
- É suportado *Enhanced Double Data Rate*, sendo a DDR-SDRAM suportada até 333 Mb/s;
- São disponibilizados entre quatro e 36 Blocos Multiplicadores dedicados;
- É disponibilizada com blocos RAM com capacidade total desde 15 Kbit até aos 648 Kbit
- Pode ter entre dois e oito DCM em função da versão adotada;
- Compatível com o protocolo PCI e com vários interfaces de configuração baseados em PROM;
- Disponível em *packaging* QFP e BGA, com *footprints* dentro do mesmo tipo de package.

No caso da Nexys 2, a FPGA é um modelo XC3S1200E, apresentando as seguintes características apresentadas na tabela 2 [70]:

XC3S1200E – Package FG320		
Número de portas do sistema		1200K
Células Lógicas Equivalentes		19.512
CLB (4 Slice por CLB)		2.168
Slice		8.672
Disposição Matricial:	Linhas	60
	Colunas	46
RAM Distribuída		136 Kbit
Blocos RAM		504 Kbit
Blocos Multiplicadores Dedicados		28
DCMS		8
I/O de Utilizador		250
Pares Diferenciais de I/O		99

Tabela 2 - Recursos da FPGA Xilinx Spartan-3E presente na Nexys 2

3.1.1.2 – Periféricos relevantes

Esta placa tem presente um oscilador que gera o sinal de relógio que alimenta o FPGA, neste caso a oscilar a 50 MHz.

Para armazenamento de grandes quantidades de dados podem ser encontradas memórias externas, ambas com uma capacidade de 128 Mbit, do tipo Flash ROM e do tipo *Synchronous Dynamic Random-Access Memory* (SDRAM).

Caso se pretenda utilizar esta placa como um sistema *standalone* estão presentes diversas ligações úteis como uma porta *Personal System/2* (PS/2), uma porta *Video Graphics Array* (VGA), uma porta *Recommended Standard 232* (RS232), vulgarmente denominada porta série, e uma porta USB2.0. Esta última é particularmente relevante porque, para além de fornecer energia à placa, permite a programação do FPGA ou da *Xilinx Platform Flash ROM* emulando um cabo JTAG.

Para I/O de sinais simples do utilizador *debugging* rápido estão presentes na face da placa oito Light Emitting Diodes (LED) de cor verde, quatro botões de pressão, oito botões de deslize e quatro *displays* de sete segmentos de cor vermelha.

Os pinos de I/O de utilizador estão acessíveis através de quatro conectores *Peripheral Module* (PMOD) de 2x6 pinos e a um conector de alto débito Hirose FX2 de 43 pinos [71].

3.1.2 – Digilent Atlys

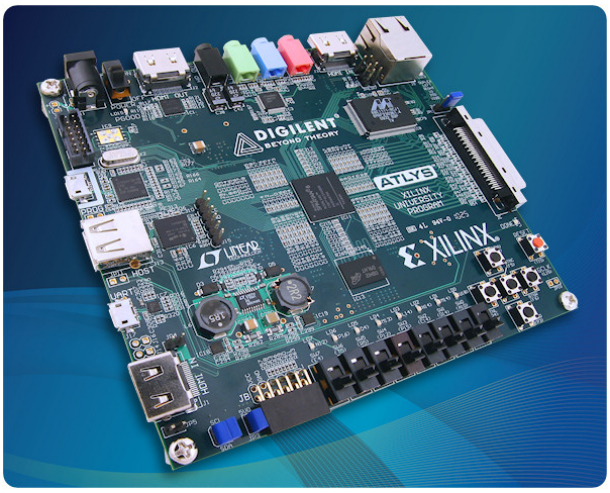

Figura 31 - Placa de prototipagem Digilent Atlys [72]

Esta placa baseia-se na FPGA Spartan-6, um IC de gama média lançado em 2009. De uma geração posterior à geração Spartan-3E apresenta um leque de características e um desempenho consideravelmente superior. Num cenário similar ao já observado para a Nexys 2, a estrutura da placa coloca a FPGA ao centro rodeando-a de outros IC relativos a periféricos e na fronteira do PCB diversos conectores de I/O, como pode ser observado na figura 31.

3.1.2.1 – FPGA Xilinx XC6SLX45 (Spartan-6 LX45)

Representando novamente a gama de FPGA da Xilinx para aplicações de elevado volume e custo por chip reduzido, a família Spartan-6 é constituída por 13 elementos, abrangendo soluções entre as 3840 e as 147443 células lógicas. Representa uma grande evolução em relação à geração Spartan-3E no que concerne o consumo energético, graças a funcionalidades de hibernação, suspensão e uma tensão de alimentação reduzida de 1,2 Volts. Apresentou também uma evolução na redução de custos e no aumento do desempenho por *chip*, dada a tecnologia de construção de 45 nanómetros.

A Spartan-6 está dividida em duas subfamílias sendo o elemento diferenciador a presença de blocos de conectividade série de alto débito numa das famílias (séries cujo nome termina em T). Este IC apresenta-se com uma lógica baseada em LUT de 6 entradas, um número superior de blocos RAM, blocos avançados para a realização de operações aritméticas DSP48A1, controladores DDR-SDRAM e blocos avançados de controlo de sinais.

As LUT de 6 entradas, registadas por dois Flip-Flops, estão desenhadas para a melhoria do desempenho, a redução do consumo energético e a aceleração dos processos que beneficiem com *pipelining*.

Quanto aos blocos DSP48A1, apresentam alta performance na realização de operações de adição, multiplicação 18 x 18 bits e MAC de 48 bits. Como o nome indica estão orientadas para a realização de operações de Processamento Digital de Sinal, oferecendo para esse efeito a possibilidade de pré-adição, essencial à realização de filtros digitais, e a capacidade de

pipelining e *cascading*, estando portanto ligadas entre si para transporte de resultados ou extensão dos limites individuais no cálculo de valores de dimensão superior à suportada por cada bloco individualmente.

Os blocos dedicados controladores de memória suportam SDRAM DDR, DDR2 e DDR3, para débitos de até 800 Mb/s, e são estruturados com múltiplas portas com FIFO independentes no sentido de reduzir eventuais problemas de *timing* das implementações.

Para controlo dos sinais de relógio existem blocos constituídos por dois elementos DCM e um elemento PLL, destinados à síntese flexível e precisa de frequências de relógio, com baixo ruído, baixo *skew* e baixa distorção.

Os seus bancos de I/O apresentam maior flexibilidade e compatibilidade que os bancos estudados para a Spartan-3E, com uma seleção mais abrangente de interfaces. São permitidas mais opções de tensão de sinalização, a seleção da corrente de *drive*, a utilização de periféricos que usam protocolos diferentes com facilidade e a variação do *slew rate* de sinais para garantir a sua integridade.

No caso da utilização de um IC que possua os blocos de conectividade série opcionais, interfaces de alto-débito como *Serial Advanced Technology Attachment* (SATA), Ethernet 1Gbit, *Ethernet/Gigabit-capable Passive Optical Network* (EPON/GPON) ou DisplayPort podem ser implementados usando os FPGA desta família.

Apresenta-se na tabela 3 as características do modelo particular implementado na Atlys [73].

XC6SLX45 – Package CSG324		
Células Lógicas		43661
CLB	Slices	6822
	Flip-Flops	54576
	RAM Distribuída	401 Kbit
Blocos DSP48A1		58
Blocos RAM 18 Kbit		116 (Total 2088 Kbit)
CMT (2 DCM+1 PLL por CMT)		4
I/O de Utilizador em 4 Bancos		218
Pares Diferenciais de I/O		92

Tabela 3 - Recuros da FPGA Xilinx Spartan-6 presente na Atlys

3.1.2.2 – Periféricos relevantes

Comece-se por abordar o oscilador que gera o sinal de relógio que alimenta o FPGA, no caso da Atlys funcionando a 100 MHz.

Para cobrir os requisitos de armazenamento de grandes volumes de dados podem ser encontradas novamente duas memórias externas, a primeira do tipo DDR2 SDRAM com uma capacidade de 1 Gbit e a segunda do tipo QuadSPI Flash ROM com uma capacidade de 128 Mbit.

Para a realização de sistemas *standalone* podem encontrar-se ligações USB 2.0, porta Ethernet 1 Gbit, diversas ligações para entrada e saída HDMI e ligações de áudio (entrada, saída, microfone, auscultadores).

No caso da Atlys as ligações USB também são multifacetadas, estando disponíveis três portas para objetivos diferentes. É possível usar uma porta para a configuração do FPGA assim como para I/O; uma segunda porta USB pode operar como *host* para utilização com um rato ou um teclado; uma terceira porta providencia um mecanismo de *Universal Asynchronous Receiver-Transmitter* (UART).

Estão disponíveis serviços de monitorização do consumo energético em tempo real associados aos vários rails de alimentação, acessíveis durante a operação da placa. Para I/O rápido e *debugging* estão presentes na face da placa oito LED de cor verde, seis botões de pressão e oito botões de deslize.

Finalmente, no sentido de fornecer ligações de uso geral ao utilizador, está disponível um conector PMOD de 2x6 pinos e a um conector de alto débito *Very High-Density Cable Interconnect* (VHDCI) de 68 pinos [74].

3.1.2.2.1 – Exar UART

A ligação da porta USB-UART presente na Atlys a um PC não instala automaticamente os drivers necessários ao funcionamento da UART de origem Exar. Essa instalação indispensável deverá ser realizada pelo utilizador seguindo os seguintes passos:

- Realizar o *download* dos *drivers* na página EXAR para o dispositivo XR21V1410 (secção “Documents” -> “Software Drivers”) [75];
- Descompactar o ficheiro “ZIP” com os drivers num local adequado;
- Aceder ao “Device Manager” (“Properties” de “Computer” -> “Manage” -> “Device Manager”);
- Se os *drivers* não tiverem sido instalados previamente, o *hardware* deverá surgir com uma simbologia adequada ao seu estado (ícone com ponto de exclamação amarelo); aceda-se a “Properties” do dispositivo;
- Na nova caixa “Properties” que se abre escolha-se “Driver” -> “Update Driver” -> “Browse My Computer For Driver Software” -> “Browse”
- Nesta fase escolha-se o driver adequado à versão do Windows instalada (32 ou 64 bits) e proceda-se à instalação escolhendo “Next”;
- Se a instalação for bem-sucedida, bastará escolher “Close”; este procedimento estará concluído.

3.2 – Ferramentas de apoio

Consideradas as placas de prototipagem utilizadas proceder-se-á à apresentação das ferramentas de apoio que as tem como alvo assim como outro *software* acessório. Para cada aplicação onde tal seja relevante mencionar-se-ão as configurações indispensáveis à obtenção de resultados.

3.2.1 – Xilinx *Design Tools*

Este pacote integrado desenvolvido pela Xilinx e em constante evolução preenche todas as necessidades de criação de projetos de implementação nos FPGA e CPLD que produz. É constituído por aplicações destinadas a cobrir os dois fluxos de projeto para FPGA já abordados na secção 2.2.5, tendo sido utilizadas a aplicação Integrated Software Environment (ISE) para o primeiro e a Embedded Development Kit (EDK) para o segundo [76].

São estas aplicações que apresentaremos de forma breve e os procedimentos e configurações necessários de uma forma mais extensa.

3.2.1.1 – *Integrated Software Environment* (ISE) - *Project Navigator*

Esta aplicação integra num *Graphical User Interface* (GUI) todas as ferramentas que permitem o desenvolvimento, programação e verificação de projeto em FPGA Xilinx. Permite esse desenvolvimento nas linguagens VHDL ou Verilog ou através de esquemático, disponibilizando ferramentas de verificação de código, ferramentas gráficas para instanciação fácil das primitivas e bibliotecas associadas a cada dispositivo produzido pela marca, ferramentas de mapeamento, encaminhamento e programação do projeto no IC. A versão utilizada para a realização dos projetos foi a 14.2.

A criação de um novo projeto passa pelos procedimentos que passaremos a descrever. Ilustraremos o processo para a placa Atlys mencionando onde seja relevante quais as configurações necessárias para a configuração da placa Nexys 2.

A figura 32 mostra o GUI que está disponível ao utilizador. Para dar início ao novo projeto clique-se em “File” e de seguida em “New Project…”.

Surge a caixa de diálogo apresentada na figura 33. Nesta caixa de diálogo escolha-se o nome do projeto e a sua localização. Recomendam-se nomes e caminhos sem espaços e de dimensão reduzida, por forma a reduzir a probabilidade de erros nos procedimentos seguintes.

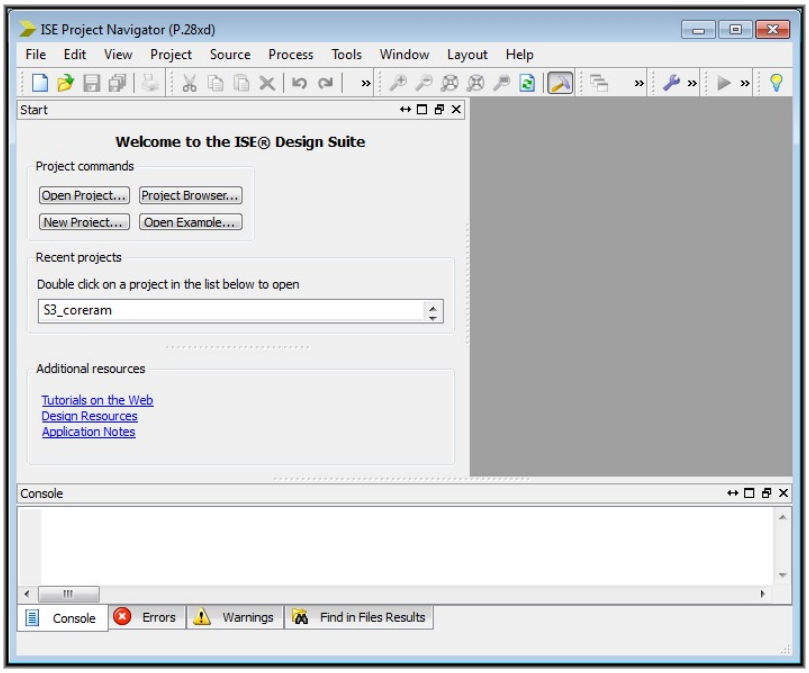

Figura 32 - Novo projeto ISE - GUI da aplicação ISE *Project Navigator*

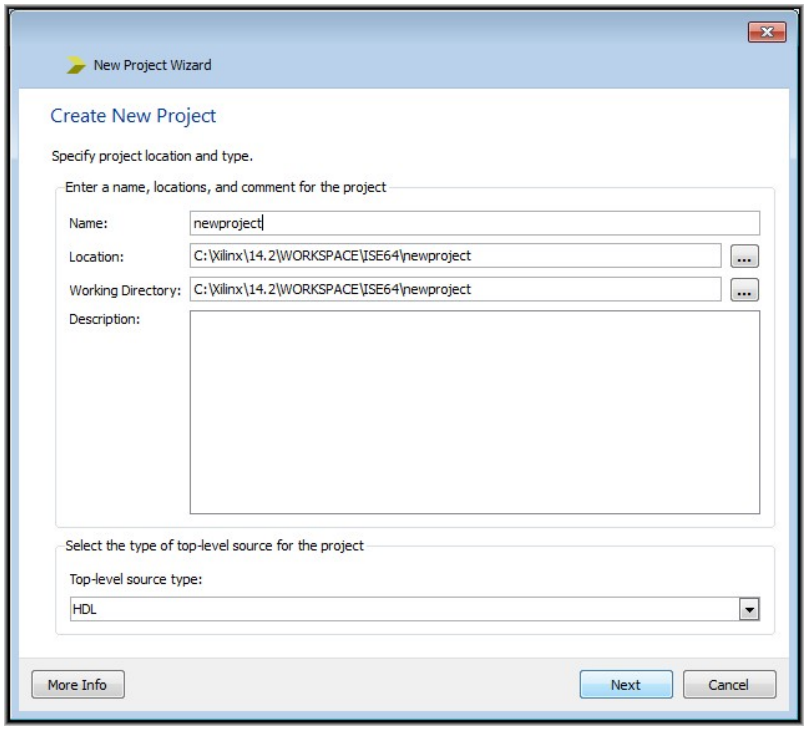

Figura 33 - Configuração do nome e da pasta de armazenamento

Escolha-se a linguagem descritiva de *hardware* pretendida, neste caso "HDL", e clique-se "Next".

A caixa de diálogo seguinte (figura 34) apresenta ao utilizador as várias configurações disponíveis para o projeto. No caso da Atlys escolha-se a família de FPGA "Spartan6", dispositivo "XC6SLX45", *package* "CSG324" e velocidade "-3", características disponibilizadas pela Digilent [77]. Para o caso da Nexys 2 escolha-se a família de FPGA "Spartan3E",

dispositivo “XC3S1200E”, *package* “FG320” e velocidade “-5” [78]. Escolha-se a ferramenta de síntese “XST (VHDL/Verilog)”, o simulador “ISim (VHDL/Verilog)” e a linguagem pretendida “VHDL”. Escolha-se finalmente “Store all values” e “VHDL-93”. Clique-se em “Next”.

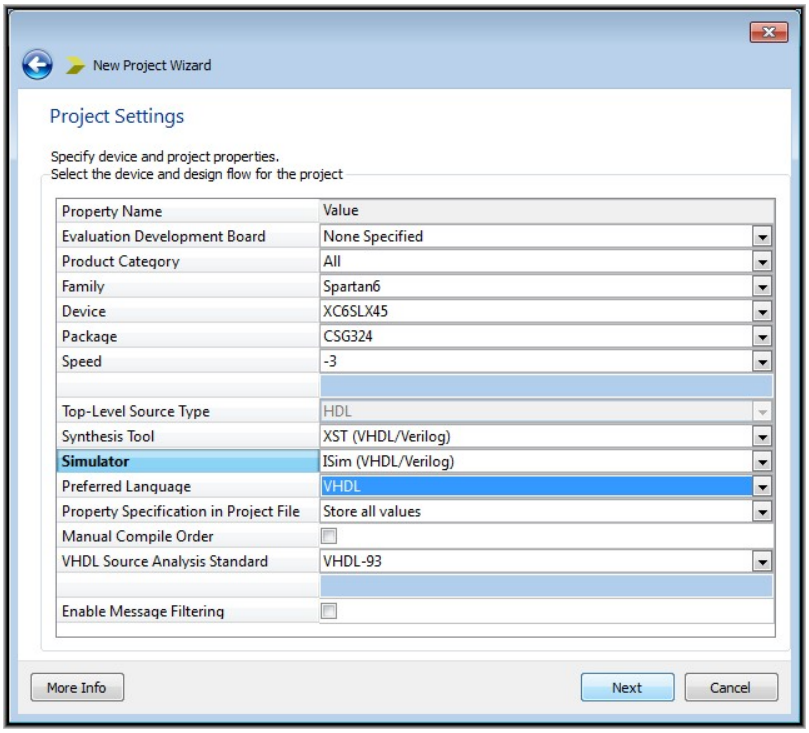

Figura 34 - Configuração do dispositivo alvo

É apresentado um resumo das escolhas realizadas na caixa de diálogo “Project Summary”. Se a configuração estiver correta, clique-se em “Finish”. A configuração de *hardware* fica assim concluída.

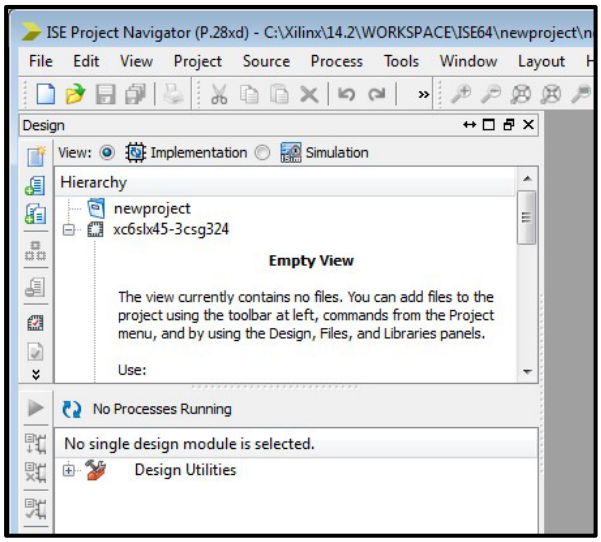

Figura 35 - Verificação da escolha de dispositivo

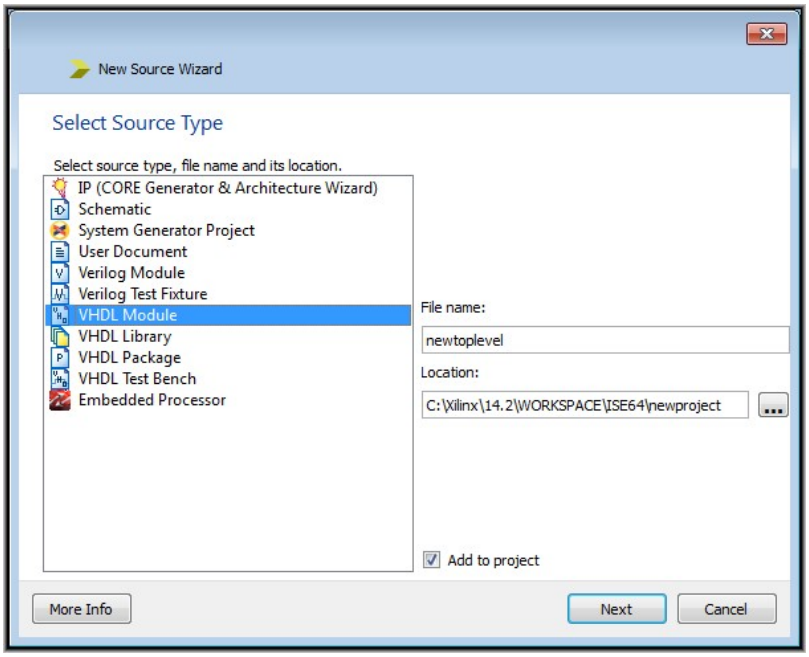

Figura 36 - Introdução de novo módulo VHDL – Escolha do nome

A caixa de diálogo fecha e é apresentado o GUI principal da aplicação ISE - *Project Navigator*. Deverá esta presente na vista "Implementation", caixa "Hierarchy" o *hardware* configurado nos passos anteriores, como apresentado na figura 35. Nesta fase podemos adicionar fontes ao nosso projeto, com o código em linguagem descritiva de *hardware* que realize a implementação pretendida. Clique-se em "Project" e de seguida em "New Source".

A figura 36 apresenta a caixa de diálogo para a adição de nova fonte ao projeto. Neste caso adicione-se uma fonte no topo da hierarquia que ligue vários blocos. No exemplo, escolheu-se "newtoplevel" como o nome e "VHDL Module" como tipo de fonte. Neste caso mantenha-se "Add to project" ativo e clique-se "Next".

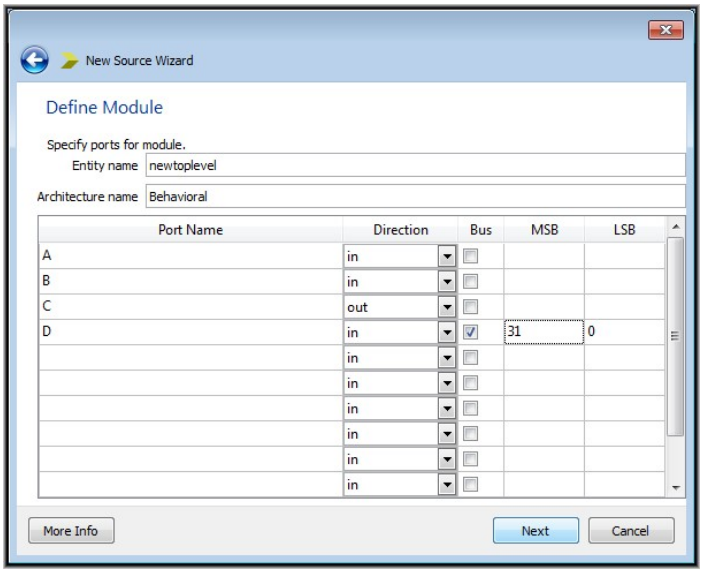

Figura 37 - Configuração dos portos

A próxima etapa compreende a escolha dos portos da nova fonte do tipo módulo VHDL. Através de uma tabela configurável, como a apresentada na figura 37, é possível a introdução rápida dos sinais de entrada e de saída pretendidos para o módulo, configurando a sua direção e, caso seja necessário, a dimensão do canal em bits. Concluída a configuração clique-se em "Next".

A etapa de conclusão apresente um sumário das escolhas realizadas. Caso a configuração esteja correta, clique-se em "Finish" para voltar ao GUI principal do Project Navigator.

Como se pode verificar na figura 38 o módulo acabado de criar é movido para a área "Hierarchy", ficando associado à FPGA alvo do projeto. Fazendo duplo clique no ficheiro, "newtoplevel – Behavioral (newtoplevel.vhd)" no nosso caso, este é aberto para edição, apresentando de imediato os portos configurados. Nesta fase pode escrever-se código através do editor e criar a nova implementação.

Tal como introduzido previamente os IP *Cores* permitem a introdução de funcionalidades das FPGA mediante uma configuração simples. Quer isto dizer que como alternativa à escrita do modelo comportamental que descreve detalhadamente o comportamento de determinado bloco lógico é possível usar um bloco previamente criado, disponibilizado com a aplicação. No caso do Project Navigator o processo é apoiado por caixas de diálogo que permitem a configuração passo a passo. Para efeito de exemplo crie-se um bloco que sintetiza um sinal de relógio a uma frequência de 250 MHz. Para isso clique-se novamente em "Project" e de seguida em "New Source…". Escolha-se a opção "IP (CORE Generator & Architecture Wizard)", atribua-se um nome ao novo núcleo, no caso em exemplo "newcoreclk", e clique-se em "Next". O sistema irá apresentar o diálogo da figura 39.

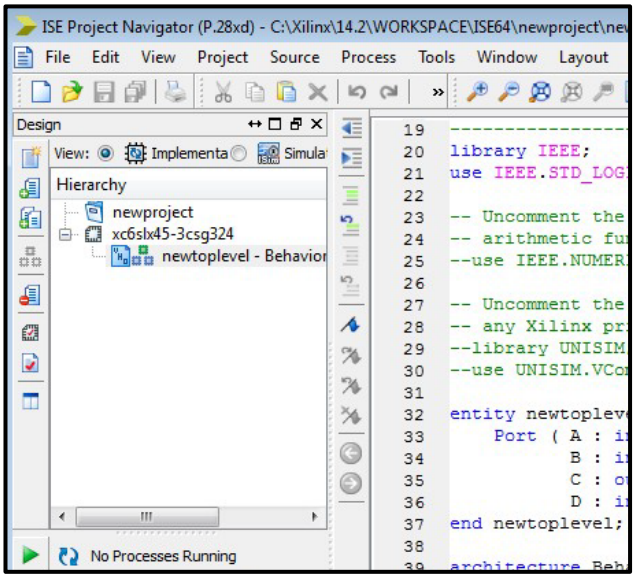

Figura 38 - Verificação da entrada do módulo para o topo da hierarquia

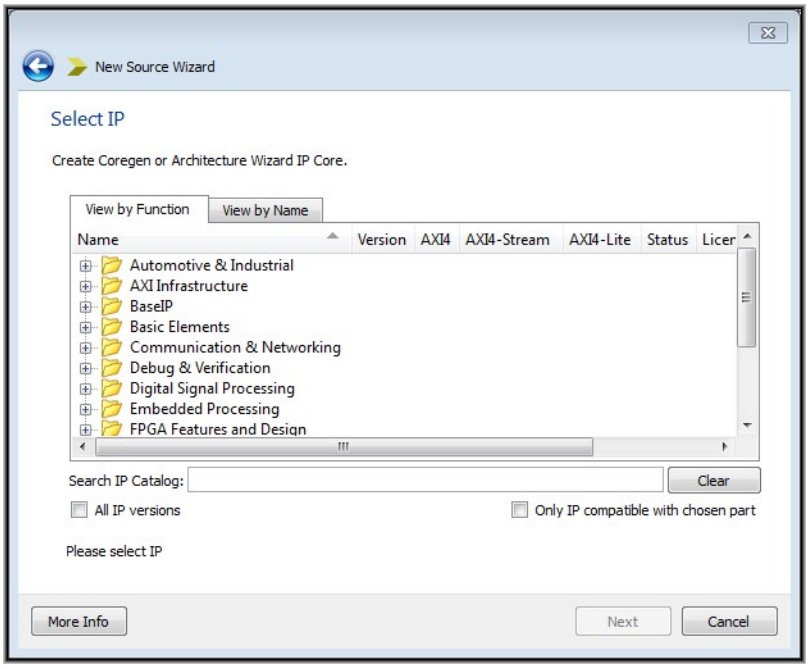

Figura 39 - Introdução de IP CORE – Bibliotecas

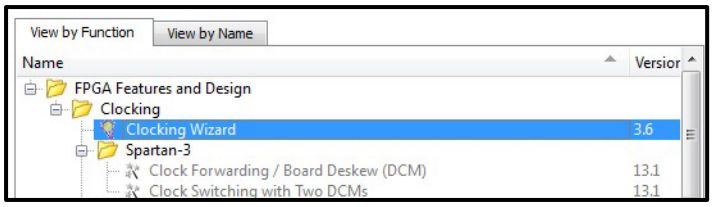

Figura 40 - Filtragem de resultados

É apresentada a lista de núcleos disponíveis, cobrindo diversos mercados e funcionalidades. Como o número de núcleos é grande, filtre-se a escolha introduzindo na área "Search IP Catalog" a palavra "clock". A filtragem apresenta diversos elementos coincidentes com a pesquisa, como se pode ver na figura 40. Repare-se que muitos núcleos surgem em tom cinza e não permitem seleção: trata-se dos núcleos que não são compatíveis com a FPGA selecionada. Escolha-se "Clocking Wizard" e clique-se "Next".

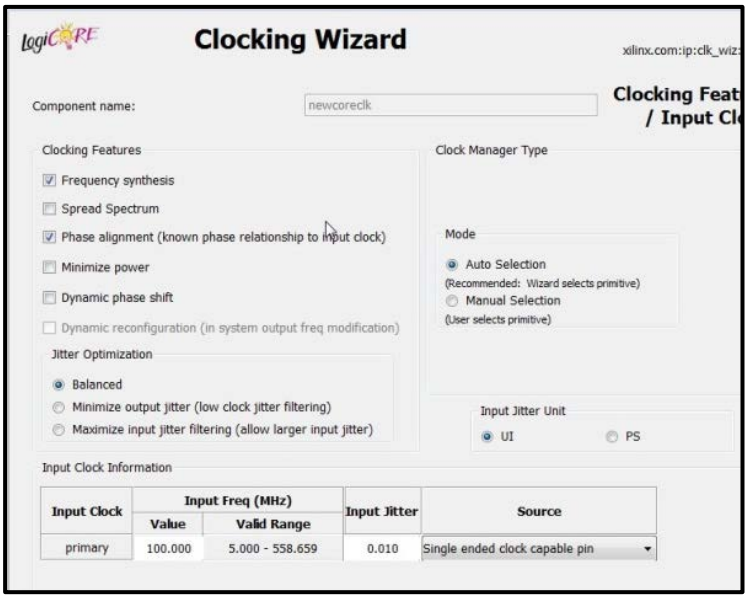

Figura 41 - Configuração 1

Na nova caixa de sumário clique-se "Finish". Somos transportados para a caixa de diálogo "CORE Generator". Esta nova caixa de diálogo, apreciável na figura 41, permite a configuração por etapas do gerador de sinal de relógio pretendido. Para efeitos de exemplo, escolha-se a síntese da frequência de 250 MHz. O sistema permite a escolha do valor de oscilação do relógio de entrada em "Input Clock" na página 1 da caixa de diálogo, coincidindo com o oscilador da placa Atlys. Clique-se em "Next".

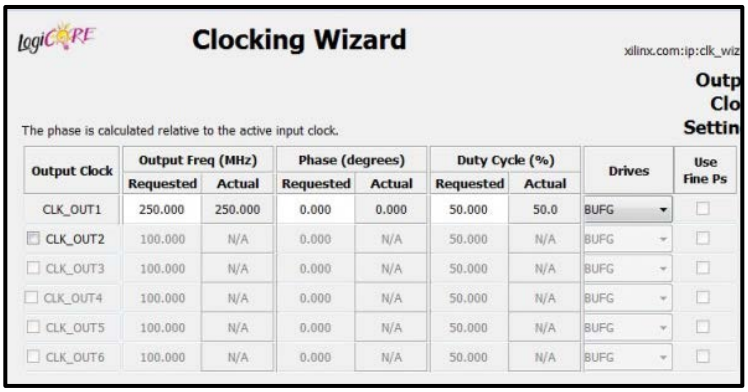

Figura 42 - Configuração 2

A página seguinte permite a escolha efetiva da frequência ou frequências a sintetizar. No nosso exemplo pretende-se uma saída a 250 MHz que deverá ser inscrita em "Requested Output Freq (MHz)". É possível ainda a configuração do desfasamento em relação ao sinal de entrada do bloco de síntese de sinal de relógio e o *duty cycle* do sinal de saída como pode ser visto na figura 42. Clique-se em "Next".

Figura 43 - Configuração 3

A figura 43 apresenta os diversos sinais de controlo opcionais selecionáveis. Escolha-se o sinal "Reset", que permite o reinício do bloco, e o sinal "Locked", que sinaliza a aquisição do sinal de relógio do oscilador presente na placa. Mantenha-se a fonte de *feedback* como "Automatic control on-chip" e clique-se em "Generate" para concluir a configuração. O bloco irá criar agora os diversos ficheiros de instanciação com as configurações introduzidas.

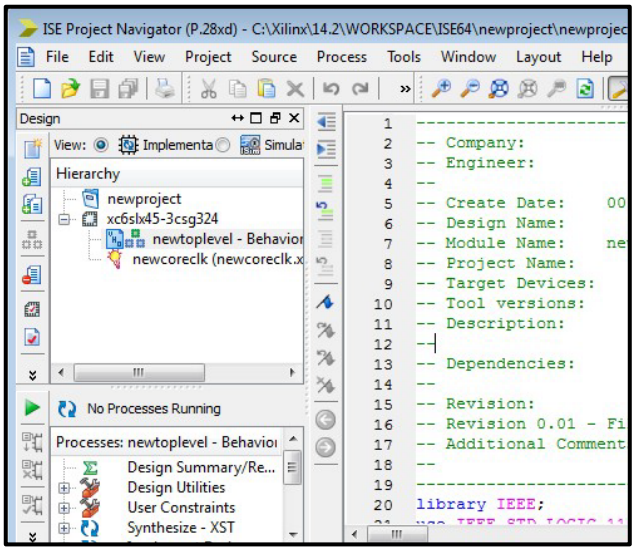

Figura 44 - Verificação da introdução do IP *Core*

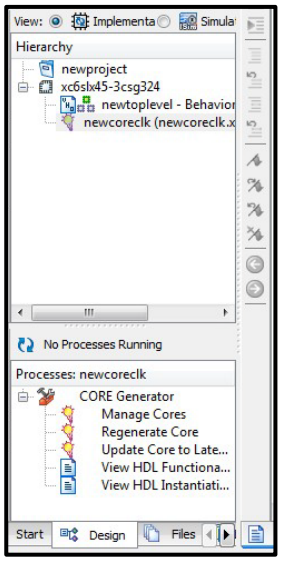

Figura 45 - Visualização do modelo de instanciação

Quando esse processo acabar o novo núcleo estará disponível na área “Hierarchy”. No entanto repare-se, por análise da figura 44, que não se encontra dependente do módulo VHDL no topo da hierarquia “newtoplevel”. Como queremos utilizar o sinal de relógio acabado de criar nesse módulo é necessário introduzir informação do núcleo criado no módulo no topo da hierarquia.

Clique-se uma vez em “newcoreclk”, o núcleo criado. Na área de “Processes” surgem diversas opções relacionadas com o núcleo. Clique-se em “View HDL Instantiation Template”, como ilustrado na figura 45, e sigam-se as indicações simples incluídas nesse ficheiro, que implicam copiar e colar o código que irá instanciar o núcleo no nosso projeto desse *template* para o módulo VHDL. Se a introdução for bem-sucedida, assim que se salvar o módulo VHDL editado com o *template* de instanciação deverá ser visível a mudança do núcleo criado para uma posição de dependência. Este resultado é ilustrado na figura 46.

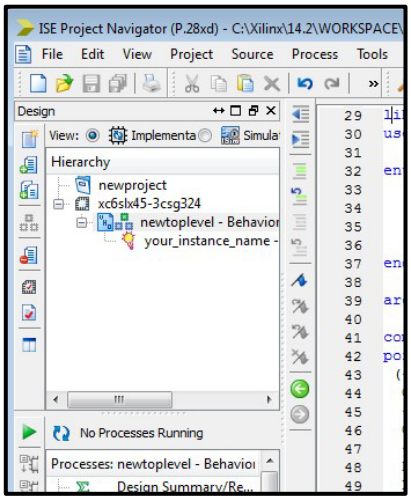

Figura 46 - Verificação da hierarquia do projeto

Após o desenho de todo o projeto, no qual poderá ser necessário introduzir módulos VHDL e IP *Cores* adicionais surgirá o momento de implementar de facto o hardware na FPGA.

Antes desse processo é preciso adicionar ao sistema um ficheiro que transmita ao projeto qual a localização dos diversos sinais provenientes de conectores ou periféricos, mapeando-os nos diversos pinos de I/O da FPGA. Esses ficheiros do tipo *User Constraint File* (UCF), usualmente disponibilizados pelos *fabric*antes das placas de prototipagem, podem ser adicionados ao projeto clicando "Project" e de seguida "Add Source…". Após encontrar o ficheiro pretendido clique-se "Open" e verifique-se a abertura correta do ficheiro, como exemplificado na figura 47. Atingida esta caixa de diálogo clique-se "OK". O ficheiro deverá ser colocado numa posição de dependência do módulo no topo da hierarquia, como visível na figura 48.

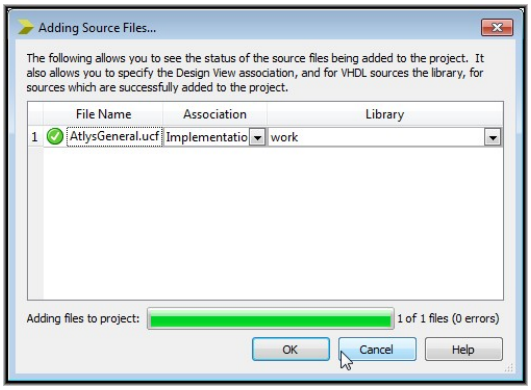

Figura 47 - Adição de módulo ao projeto (UCF)

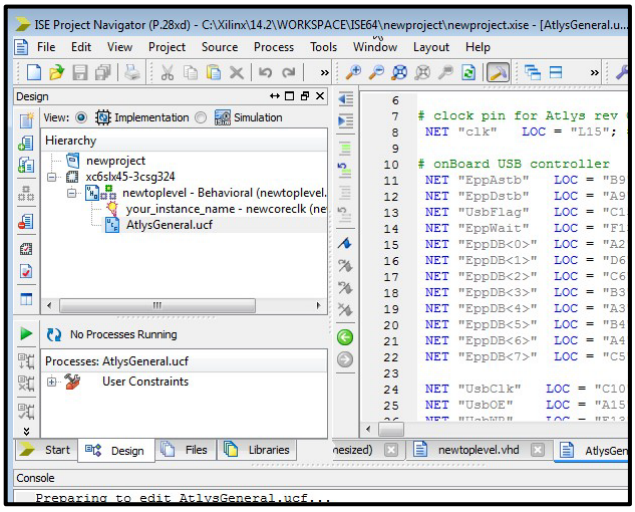

Figura 48 - Verificação da hierarquia do projeto

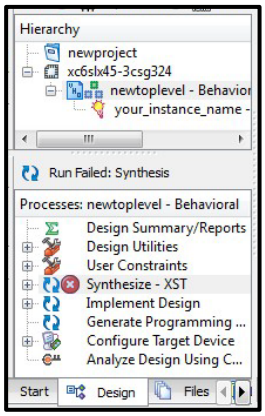

Figura 49 - Geração do ficheiro de programação

Para criar o ficheiro *BitStream* que permitirá a configuração da FPGA escolha-se o módulo VHDL no topo da hierarquia e atente-se na área "Processes". Entre diversas opções apresentadas na figura 49 está presente "Generate Programming File". Clique-se duas vezes nessa opção e aguarde-se que o processo conclua. O processo irá atravessar três fases:

- A síntese do projeto criado, traduzindo a linguagem descritiva de *hardware* nos elementos físicos disponíveis na arquitetura da FPGA alvo;
- A implementação do projeto, mapeando os diversos elementos sintetizados e encaminhando todos os sinais entre eles;
- A geração do ficheiro de programação a carregar na FPGA.

Caso algo falhe, ou alguma opção possa ser identificada como perigosa para o bom funcionamento do projeto, surgirão informações nas áreas "Errors" e "Warnings" respetivamente. Caso só sejam emitidos avisos, o ficheiro de programação terá condições para ser concluído, sendo apresentados sinais de aviso amarelos na área "Processes". Caso seja emitido um erro o processo é parado e o ponto de término forçado é identificado com um símbolo de aviso vermelho.

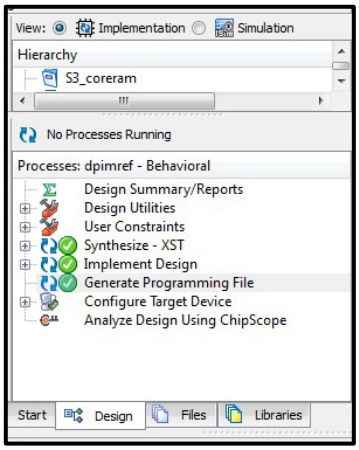

Figura 50 - Geração de ficheiro de programação bem-sucedida

Caso contrário, se o projeto cumprir os requerimentos das regras de implementação, deverá concluir com sucesso a criação do ficheiro *BitStream* surgindo sinalização verde como a exemplificada na figura 50. Fica concluída assim a apresentação da configuração de novos projetos usando a aplicação ISE - *Project Navigator*.

3.2.1.2 – *Embedded Development Kit* (EDK)

O EDK é um ambiente integrado para o design de sistemas baseados em microprocessadores embebidos, suportando as opções *Hard Core* PowerPC e *Soft Core* MicroBlaze. Providencia um ambiente gráfico completo que facilita a introdução dos IP *Cores* disponibilizados nas suas bibliotecas ou realizados por terceiros, permitindo a configuração destes de forma acessível através de *wizards*, providenciando ferramentas de criação e visualização instantânea das ligações criadas entre blocos. Fornece também um *Integrated Development Environment* (IDE) para o desenvolvimento de aplicações que usem o hardware instanciado.

Esta ferramenta divide-se portanto em dois grandes blocos, que se reportam ao *design flow* estudado para soluções *embedded*: Xilinx Platform Studio (XPS) para o desenvolvimento do BitStream relativo ao *hardware* e Software Development Kit (SDK) para o desenvolvimento do *software* a executar [79].

3.2.1.2.1 – Xilinx *Platform Studio* (XPS)

A aplicação XPS inclui um GUI e suporte de linha de comandos para o desenvolvimento de plataformas de *hardware* para aplicações de microprocessador embutido. Inclui um configurador para a obtenção de resultados de forma acessível, denominado *Base System Builder* (BSB), assim como configuradores para os vários elementos da arquitetura do sistema, canais de comunicação e periféricos.

Os passos seguintes deverão ser os primeiros a realizar quando se pretenda implementar um sistema baseado em microprocessador MicroBlaze. Irão ser apresentados de forma ilustrada para a placa Atlys e de forma textual, onde necessário, as configurações específicas à placa Nexys 2. Por motivos de compatibilidade com os elementos disponibilizados pelo fabricante que permitem a construção destes projetos utilizou-se a versão 14.1 deste *software*.

O primeiro passo a realizar é iniciar a aplicação XPS, sendo o quadro inicial aquele ilustrado pela figura 51. Escolha-se “Create New Project Using Base System Builder” diretamente na coluna “Getting Started” ou clicando na mesma opção na *tab* “File”.

A primeira caixa de diálogo deste processo permite a escolha da localização para salvar o projeto criado, o tipo de canal utilizado, o carregamento de ficheiros de configuração BSB de sessões precedentes e a entrada do diretório com o repositório de periféricos a pesquisar.

Dado que algumas funções e ferramentas disponibilizadas pela Digilent através dos seus exemplos foram criadas para uma implementação PLB, optou-se por utilizar esse tipo de

canal de ligação. A opção AXI é o estado da arte mas apresentou problemas de compatibilidade com os repositórios de periféricos, pelo que foi ignorada.

Figura 51 - Novo projeto XPS - GUI da aplicação

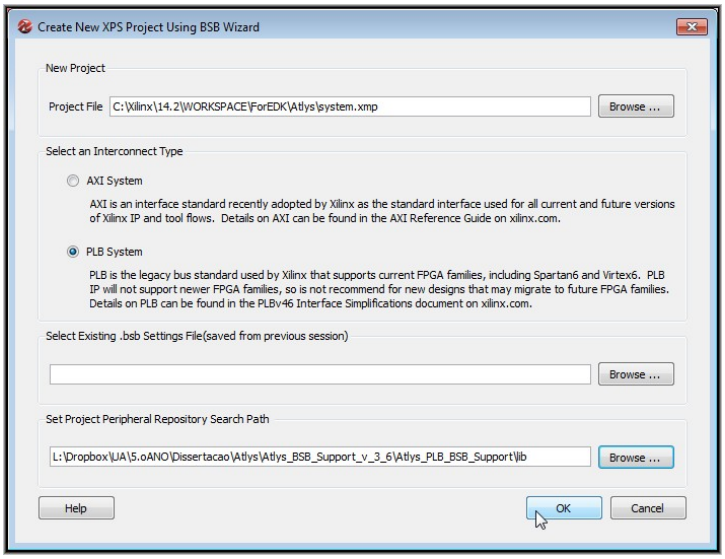

Figura 52 - Configuração de pastas para armazenamento e repositório - Atlys

Para que a placa seja reconhecida, assim como o *hardware* periférico que é possível configurar, deverá pesquisar-se pelo repositório de periféricos fornecido pela Digilent e que deverá ser descarregado previamente do seu site tanto para o caso Atlys como para o caso Nexys 2 [80][81].

Refira-se que se no caso Nexys 2 só são fornecidas bibliotecas para canais de ligação PLB, no caso Atlys são fornecidas soluções tanto PLB como AXI. Ambos os *packs* incluem ficheiros com instruções acerca da criação de projetos usando o BSB, incluindo a configuração de periféricos que não serão utilizados neste projeto.

No caso do *pack* para Atlys, o ficheiro "ZIP" deverá ser descompactado em local conveniente e o caminho fornecido ao *wizard* BSB deverá ser do tipo (ver figura 52):

"…\Atlys_BSB_Support_v_3_6\Atlys_PLB_BSB_Support\lib".

Para o pack Nexys 2 o caminho deverá ser do tipo:

"…\Nexys2_BSB_Support_v_3_0\lib".

Antes de clicar em “OK” verifique-se que nenhum dos caminhos escolhidos nesta caixa possui espaços, sendo recomendados caminhos exclusivamente alfa-numéricos com espaçamento usando under-score, preferencialmente curtos e localizados em disco.

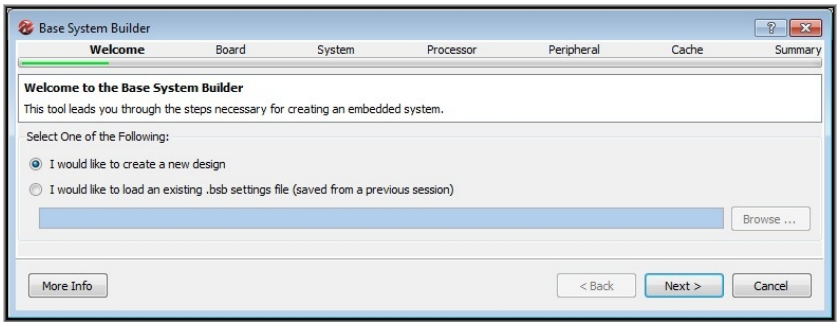

Figura 53 - Escolha do tipo de projeto

A caixa de diálogo “Welcome”, representada na figura 53, apresenta a possibilidade de carregar configurações de implementações realizadas anteriormente. Escolha-se “I would like to create a new design” e clique-se em “Next”.

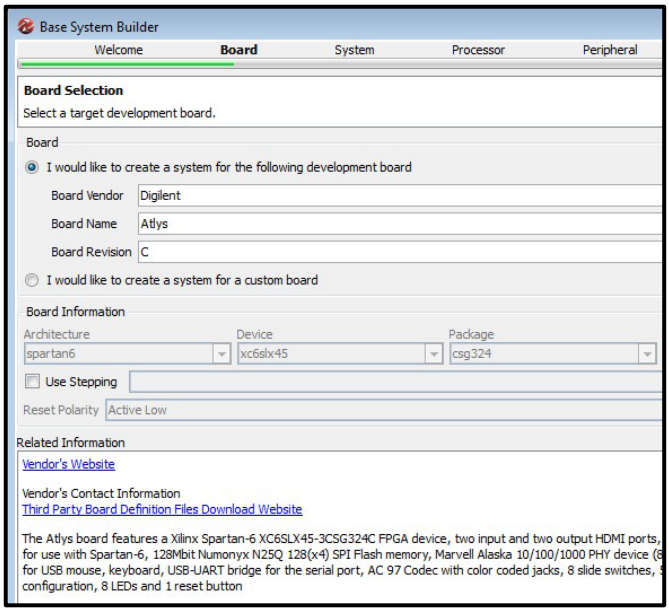

Figura 54 - Validação da placa de prototipagem

A caixa de diálogo “Board” deverá neste ponto reconhecer as placa em estudo, identificando o Fabricante, o nome da placa e a revisão dessa placa, ilustrado pela figura 54 para o caso Atlys. No caso da Nexys 2 existem duas possibilidades relacionadas com o elemento da família Spartan-3E presente na placa. No nosso caso de estudo escolheu-se “Nexys 2-1200 Board”. Clique-se em “Next”.

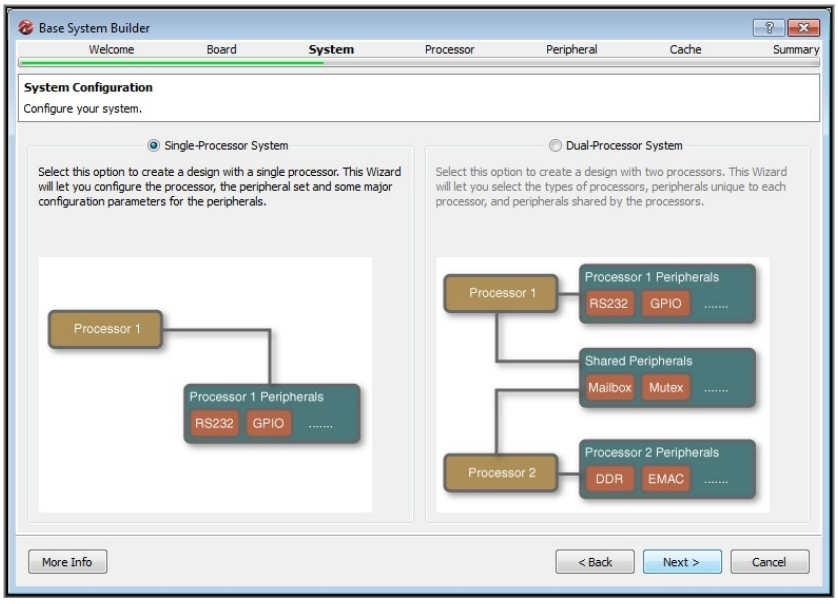

Figura 55 - Escolha do número de núcleos do MicroBlaze

A caixa de diálogo que se segue permite a escolha do número de microprocessadores pretendidos para implementação, como apresentado na figura 55. Para este estudo, um microprocessador é suficiente portanto escolha-se "Single-Processor System" e clique-se em "Next".

Seguidamente chega-se à caixa "Processor" onde é possível escolher o tipo de processador, a frequência de operação do sistema e a memória local a implementar. Dado que ambas as placas utilizadas têm FPGA embutidos sem um bloco microprocessador discreto, a única opção será "MicroBlaze".

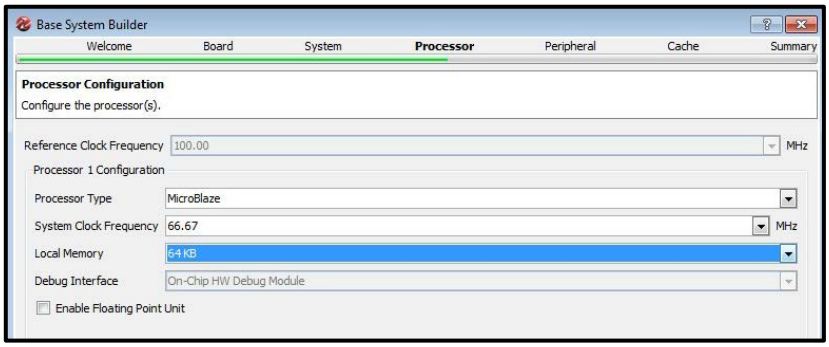

Figura 56 - Configuração do processador

No caso Atlys a frequência de operação selecionada respeita a frequência de operação utilizada nas implementações exemplo da Digilent, pelo que foi mantida a "66.67" MHz e no caso Nexys 2 a frequência de operações escolhida deverá ser "50.00" MHz, por omissão. Finalmente, escolha-se "64 KB" e "32 KB" para a configuração da dimensão da memória local de Atlys e Nexys 2 respetivamente. Estes procedimentos estão apresentados para a placa Atlys na figura 56. Clique-se em "Next".

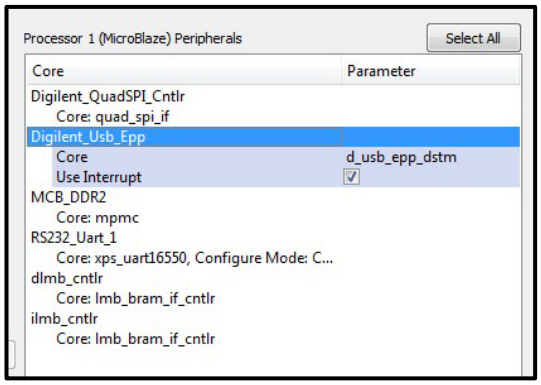

Figura 57 - Configuração de periféricos 1

A janela apresenta de seguida as opções de "Peripheral" e permite neste ponto realizar uma pré-configuração de periféricos que se pretendam na implementação.

No caso de uma configuração Atlys optou-se pelos blocos:

- "Digilent_QuadSPI_Cntlr" para utilizar a QuadSPI-Flash presente na placa;
- "Digilent_Usb_Epp" para utilizar a ligação USB em modo *Enhanced Parallel Port* (EPP), configurado com interrupções;
- "MCB_DDR2" respeitante à memória DDR2-SDRAM;
- "RS232_Uart_1" respeitante à porta UART, configurado como "xps_uart16550".

No caso da Nexys 2 realizou-se uma escolha similar, alterando as memórias externas pelos blocos "INTEL_FLASH" e "Micron_RAM", respeitantes à memória Flash e SRAM presentes na placa.

As configurações mencionadas estão patentes nas figuras 57 e 58 para a placa Atlys.

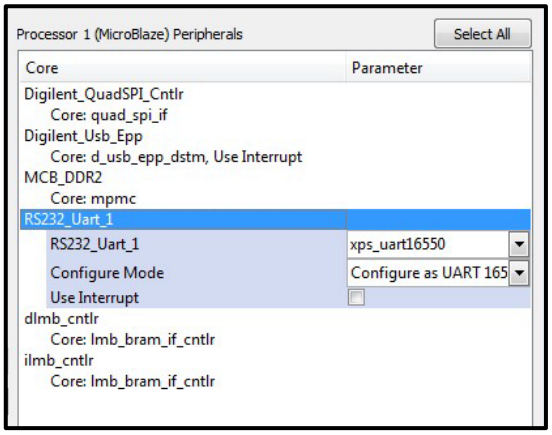

Figura 58 - Configuração de periféricos 2

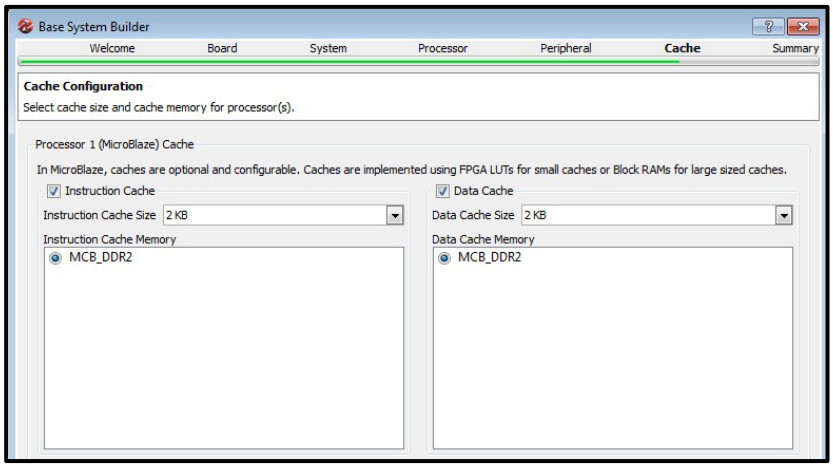

Figura 59 - Configuração da cache

A caixa seguinte, "Cache", permite a configuração da cache de dados e instruções para as memórias presentes nas placas (vide figura 59).

No caso Atlys selecione-se "2 KB" para ambos os tipos de *cache*, de dados e de instruções. No caso da Nexys 2 selecione-se, por motivos de compatibilidade com a FPGA em utilização, uma cache de "1 KB" para dados e para instruções, e a memória sobre a qual usar *cache* "Micron_RAM". Não faz sentido usar este procedimento na memória Flash pois o seu acesso é comparativamente muito mais lento. Clique-se em "Next".

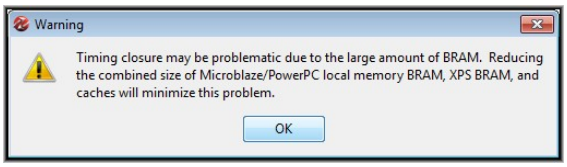

Figura 60 - Configuração da cache - Aviso comum

Uma informação poderá surgir mencionando a probabilidade de conflitos com a configuração selecionada, tal como apresentado na figura 60. Clique-se em "OK".

A caixa final "Summary", apresenta uma listagem de periféricos selecionados, o seu espaço de endereçamento e também os ficheiros relevantes relativos à configuração criada, que estão disponíveis em disco. Clique-se em "Finish".

É apresentada então a implementação escolhida, sendo possível visualizar três grandes planos de apreciação do sistema (vide figura 61):

- "Design Summary", por norma apenas relevante após a geração do BitStream de configuração da FPGA, onde é possível avaliar a ocupação dos elementos do FPGA bem como outros resultados da implementação;
- "Graphical Design View", que permite a visualização instantânea das ligações entre microprocessador, canais, periféricos e pinos/*pads* do IC;
- "System Assembly View", que apresenta através de um esquemático as ligações entre blocos ordenadas por canal, informação acerca dos interfaces, portas e endereços de mapeamento do sistema.

Selecione-se o “System Assembly View” e clique-se duas vezes sobre “microblaze_0” no *tab* “Bus Interfaces”.

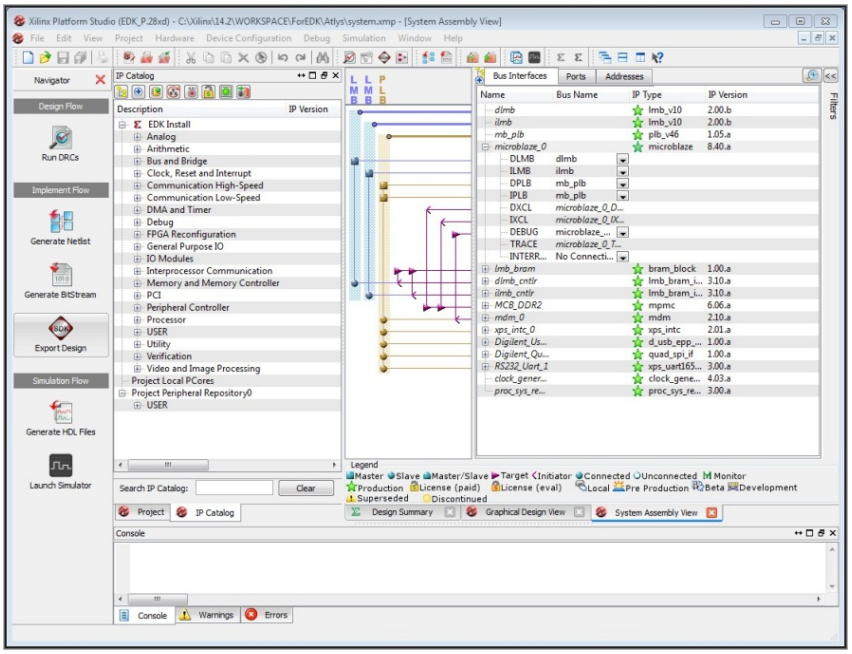

Figura 61 - Vista *System Assembly*

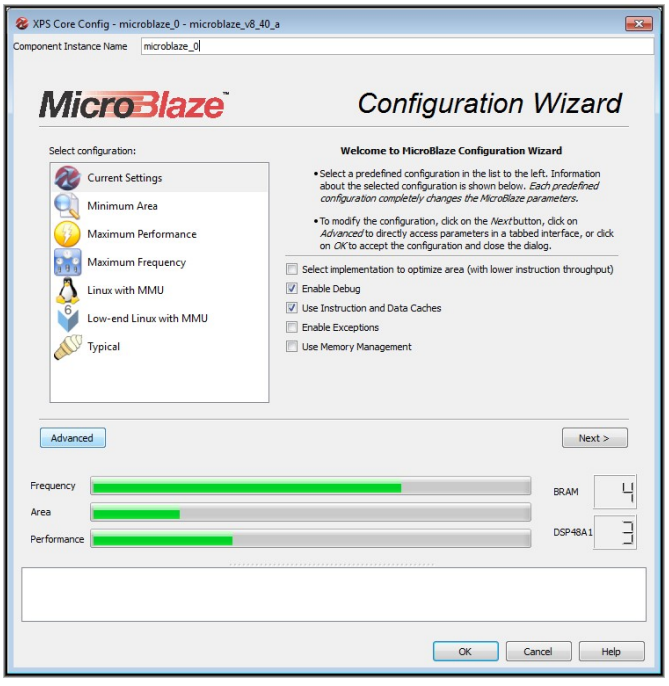

Figura 62 - Configuração MicroBlaze - Wizard

Através de um configurador é possível adaptar o microprocessador MicroBlaze às necessidades do projeto. São apresentados *templates* de configuração e informações

qualitativas associadas a cada um, permitindo estimar de imediato qual a frequência de operação, área lógica do IC ocupada e desempenho projetado. Escolha-se nesta vista apenas as opções “Enable Debug” e “Use Instruction and Data Caches”, como concretizado na figura 62. Clique-se em “Advanced”.

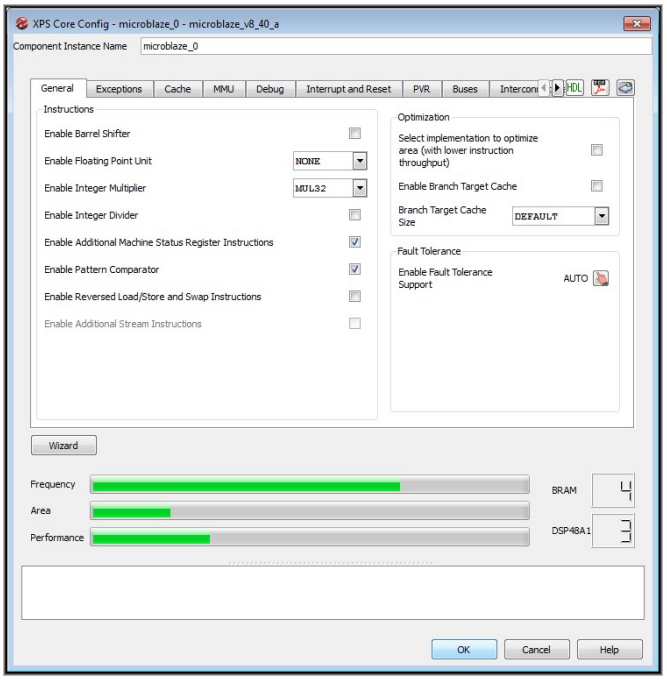

Figura 63 - Configuração MicroBlaze - Advanced

Em ambos os casos, Nexys 2 e Atlys, libertem-se as opções “Enable Barrel Shifter” e “Enable Reversed Load/Store and Swap Instructions”, mantendo ativas apenas “Enable Additional Machine Status Register Instructions” e “Enable Pattern Comparator” como configurado na figura 63. Clique-se em “OK”.

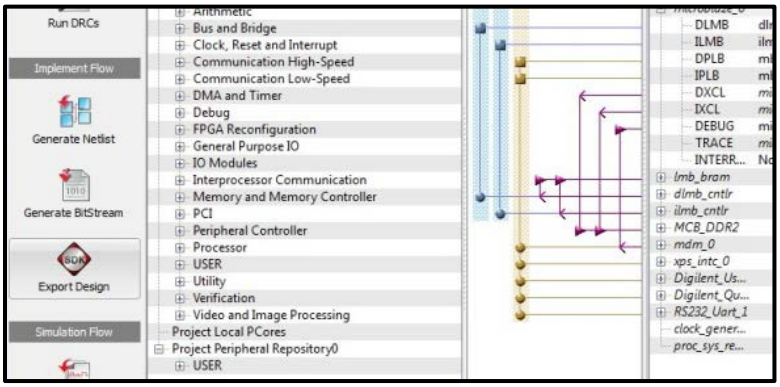

Figura 64 - Exportar projeto 1

Concluída a configuração do *hardware* a implementar, clique-se em “Export Design” na área “Navigator” à esquerda da janela principal do GUI XPS, como presente na figura 64.

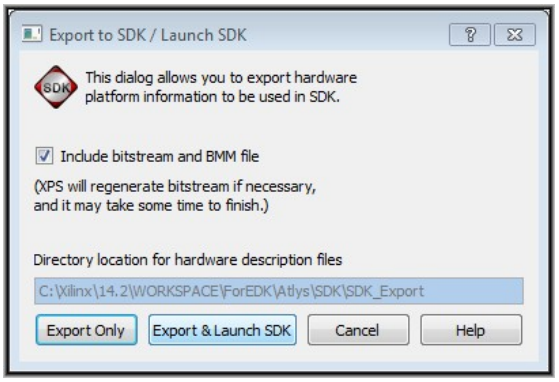

Figura 65 - Exportar projeto 2

Surgirá a caixa de diálogo da figura 65. Escolha-se "Include BitStream and BMM file" e clique-se em "Export & Launch SDK". A partir desse momento a aplicação XPS fará síntese, mapeamento e roteamento do sistema escolhido, num processo que poderá demorar bastante tempo, dependendo do PC onde esta operação for realizada. Como referência, a conclusão desta operação usando CPU Intel i7 a 2.67 GHz e 6 GByte de DDR3-SDRAM demora em média 15 minutos e usando um CPU mobile P6100 a 2 GHz e 3 GByte de DDR3-SDRAM cerca de 30 minutos, ambos os valores para a criação do *BitStream* para a Atlys. No caso da Nexys 2 o processo é bastante mais demorado, devido a um maior número de iterações na otimização do mapeamento e encaminhamento do projeto.

Quando o processo terminar a aplicação SDK abrirá automaticamente, e na sua área de trabalho estará presente o *BitStream* e informações acerca do *hardware* acabado de implementar. Os procedimentos desse ponto em diante serão abordados na secção seguinte.

3.2.1.2.2 – Software Development Kit (SDK)

Esta aplicação, com um ambiente baseado no IDE Eclipse, inclui compilador e *debugger* GNU C/C++ para a criação de aplicações em ambientes embutidos e a ferramenta Data2MEM que permite o carregamento e atualizações do *software* a executar em FPGA Xilinx. É a base de trabalho para a criação de código que use o microprocessador MicroBlaze.

Por forma a dar continuidade aos procedimentos descritos para a criação do *hardware* necessário usando a aplicação XPS, apresenta-se a aplicação SDK com dados referentes a esse processo já presentes na área de trabalho.

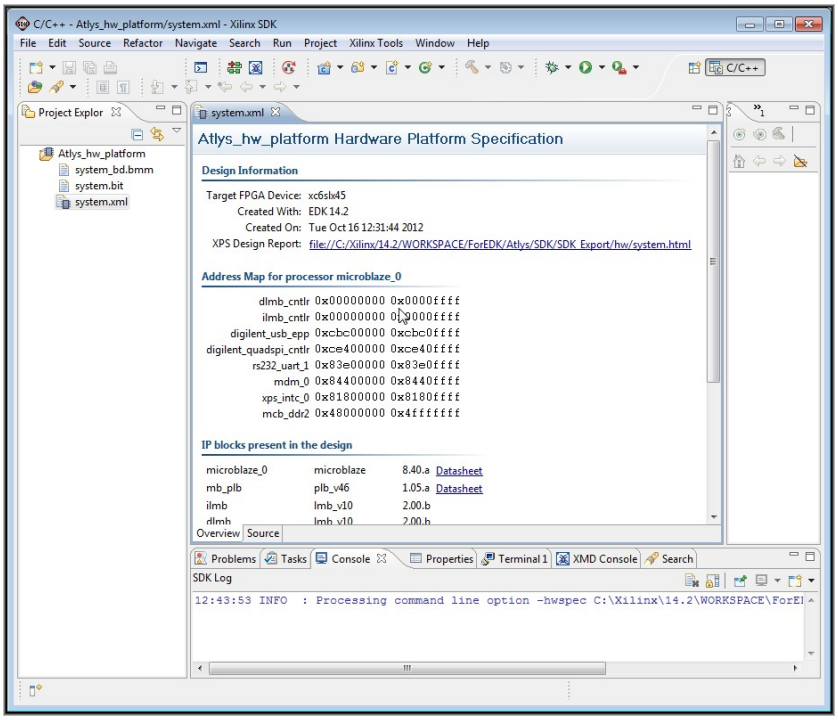

Figura 66 - Novo projeto SDK - GUI da aplicação

Repare-se por análise da figura 66 que o conjunto de informações disponibilizado no ficheiro "XML" "Hardware Plaftorm Specification" é transportado diretamente da configuração feita na aplicação XPS. Estão presentes, de acordo com as opções realizadas na conclusão desse processo, os ficheiros *BlockRAM Memory Map* (BMM) e *BitStream* (BIT).

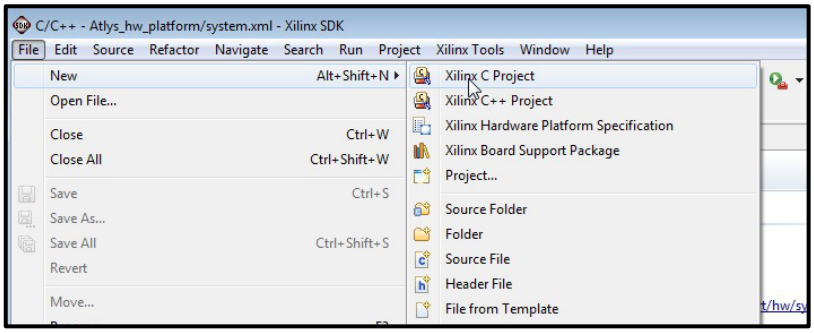

Figura 67 - Início de novo projeto C

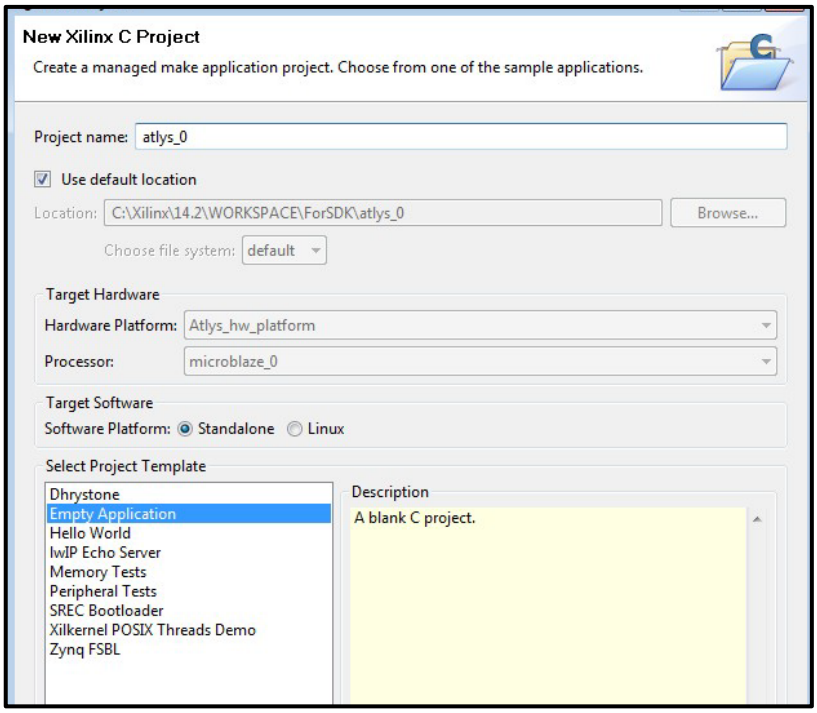

Figura 68 - Nome e tipo de módulo C

Para a criação de um novo projeto, clique-se em "File", "New" e escolha-se "Xilinx C Project" como ilustrado na figura 67. Foi esta a escolha para os nossos projetos dado que a linguagem C está presente nas bibliotecas de periféricos e exemplos de implementação Digilent, mantendo assim coerência de linguagem em todo o projeto e implementação. Nesta fase é apresentada a caixa de diálogo de novo projeto. Escolha-se um nome para o projeto, o tipo "Standalone" e um *template* "Empty Application" e clique-se em "Next" (Figura 68).

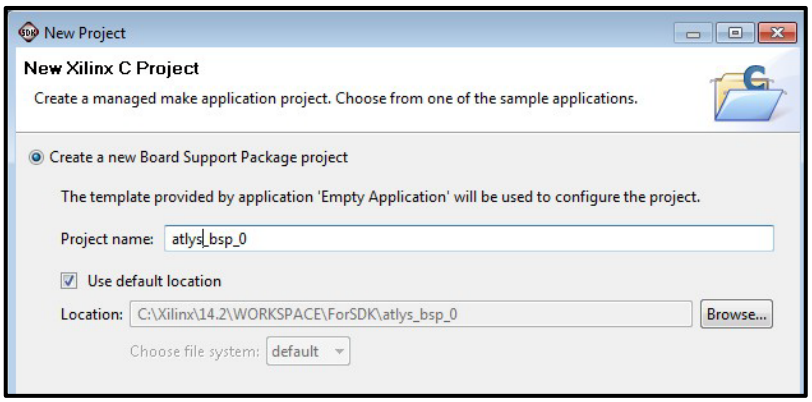

Figura 69 - Nome do novo *Board Support Package*

Escolha-se um nome para a "Board Support Package", pacote de bibliotecas e estruturas que permitirão a ligação entre o *hardware* instanciado e o projeto desenhado em C, apresentada na Figura 69. Clique-se em "Finish".

Criado o projeto vazio em C bastará iniciar novos ficheiros de código ou transportar ficheiros existentes para o projeto. O processo de criação concretiza-se clicando propriedades na pasta "src" de fontes do projeto e de seguida "New" -> "Source File".

Dê-se um nome a esse ficheiro novo, incluindo a extensão ".c" e clique-se em "Finish". Alternativamente arraste-se o ficheiro "C" de uma janela de explorador do Windows para a pasta "src", sendo criada de imediato na pasta do projeto uma cópia do ficheiro arrastado. Pode a partir deste ponto desenhar-se o projeto C sobre os ficheiros arrastados ou criados..

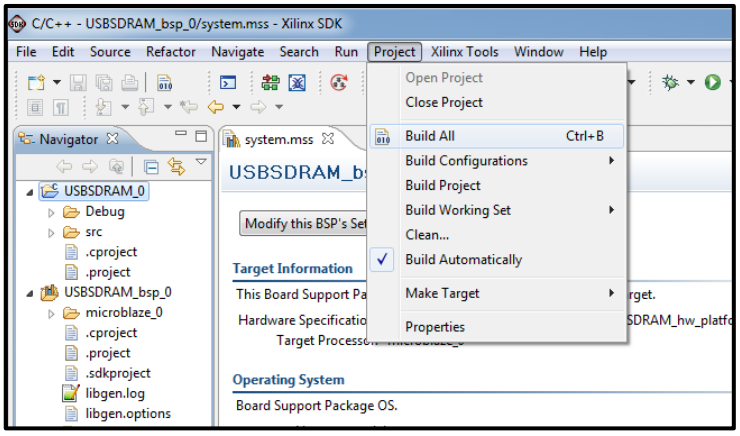

Figura 70 - *Build All*

Quando o design for concluído bastará clicar em "Project" e de seguida em "Build All", ilustrado na Figura 70. Este procedimento fará a construção do projeto, entre verificação, *linkagem* e compilação.

Dado que este procedimento é uma forma rápida e acessível de verificar o projeto realizado ao longo do processo de escrita, recomenda-se a utilização de duas combinações de teclas: "CTRL + S" salvando o projeto seguido de "CTRL + B" realizando "Build All".

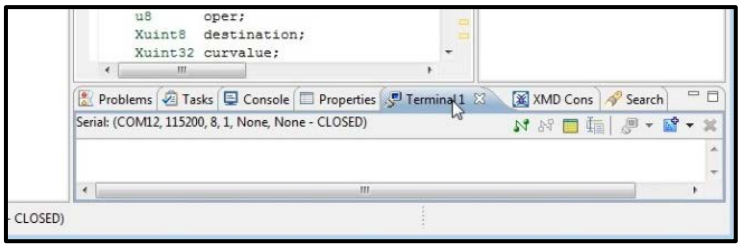

Figura 71 - Configuração do terminal/UART 1

Caso o projeto inclua um bloco UART deverá realizar-se nesta fase a sua configuração. Para tal verifique-se primeiro qual a porta "COM" utilizada pela UART, e que estará definida no "Device Manager" do ambiente Windows. Identificado o número da porta de comunicações série, por exemplo "COM12", clique-se em "Terminal" no tab da área inferior da janela da aplicação SDK, e em seguida em "Settings". A figura 71 ilustra a área do GUI onde se devem realizar estas operações.

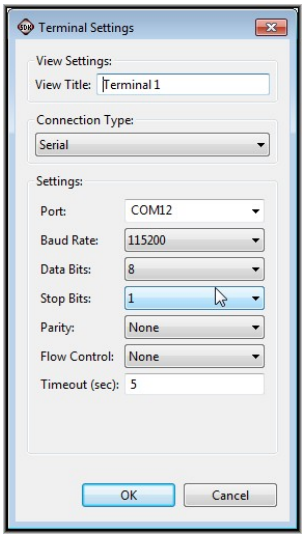

Figura 72 - Configuração do terminal/UART 2

Na nova janela, apresentada na figura 72, escolha-se o nome da ligação, o tipo de ligação "Serial", o "Baud Rate" a 115200 Kbit, "8" "Data bits", "1" "Stop bits", sem função de paridade nem controlo de fluxo e com um "Timeout" de "5" segundos. Foram estas as configurações de inicialização adotadas para os diversos projetos criados. Clique-se em "OK".

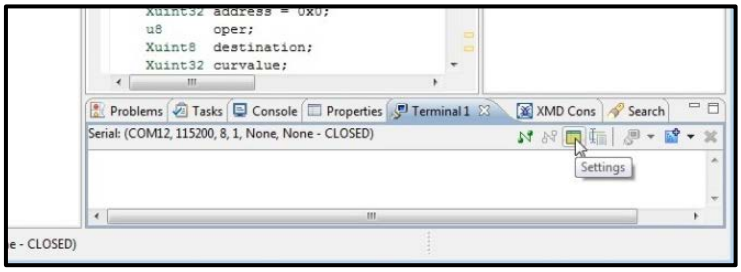

Figura 73 - Configuração do terminal/UART 3

Concluída esta configuração da UART clique-se em "Connect", o símbolo "N" de côr verde visível na figura 73 à esquerda do ponteiro do rato. O terminal apresentará erro caso algo falhe. No canto inferior esquerdo da janela da aplicação SDK surgirá uma mensagem de "CONNECTED" caso a ligação seja bem-sucedida.

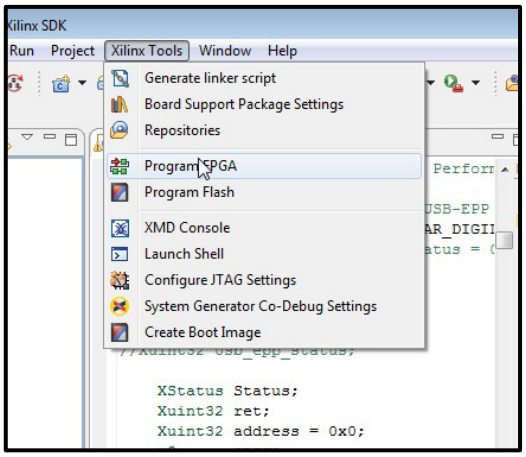

Figura 74 - Programação da FPGA com *BitStream* e ELF 1

Os passos finais implicam a programação de *hardware* e *software* compilado na FPGA. Para isso clique-se "Xilinx Tools" e de seguida "Program FPGA", como apresentado na figura 74.

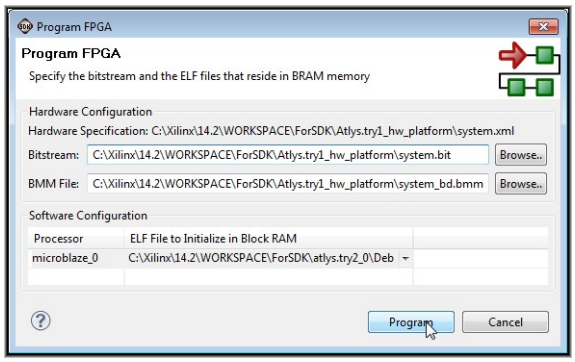

Figura 75 - Programação da FPGA com BitStream e ELF 2

Escolha-se o ficheiro "ELF" adequado selecionável em "ELF File to Initialize in Block RAM". Se apenas estiver presente a opção "bootloop" algum passo do procedimento de "Build All" terá falhado. Clique-se em "Program" (figura 76).

A programação da FPGA fica assim concluída, sendo possível comunicar com a FPGA usando a UART a partir do terminal ativado, caso esse bloco tenha sido configurado.

3.2.2 – Visual Studio 2012

Este IDE produzido pela Microsoft está desenhado para a criação de soluções como aplicações de consola ou aplicações com GUI, soluções *web*, soluções orientadas a *cloud-computing*, servidores ou até terminais móveis. Suportando várias linguagens de programação, desde C/C++/C# ou Python até Visual Basic, trata-se de um IDE ou seja reúne num só GUI acessível e descritivo os elementos e opções necessários a um desenvolvimento rápido de aplicativos. Permite ao utilizador final, por um lado, uma abstração dos procedimentos e complexidade de invocação de compilador, *linker* ou *debugger*, como por

outro a possibilidade de configuração extensa e precisa dos requisitos para cada etapa do processo de criação de soluções.

O Visual Studio 2012 foi utilizado para a criação de aplicações de consola escritas em C e os procedimentos seguidos serão descritos nos parágrafos seguintea.

Iniciada a aplicação clique-se em "File", "New" e "Project" como demonstrado na figura 76.

Na nova janela de diálogo escolha-se "Templates" na área à esquerda, "Visual C++" e de seguida "Empty Project". Dê-se um nome ao projeto e clique-se em "OK" (vide figura 77).

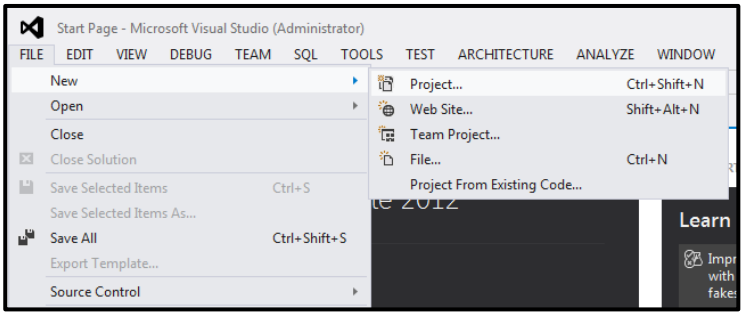

Figura 76 - Novo projeto Visual Studio - GUI da aplicação e novo projeto

Alguns projetos criados apoiam-se em bibliotecas e ficheiros de cabeçalhos disponibilizados de forma livre *online*. É possível, arrastando os ficheiros biblioteca e cabeçalho de uma janela do explorador do Windows para o plano "Solution Explorer", adicionar estes elementos ao projeto.

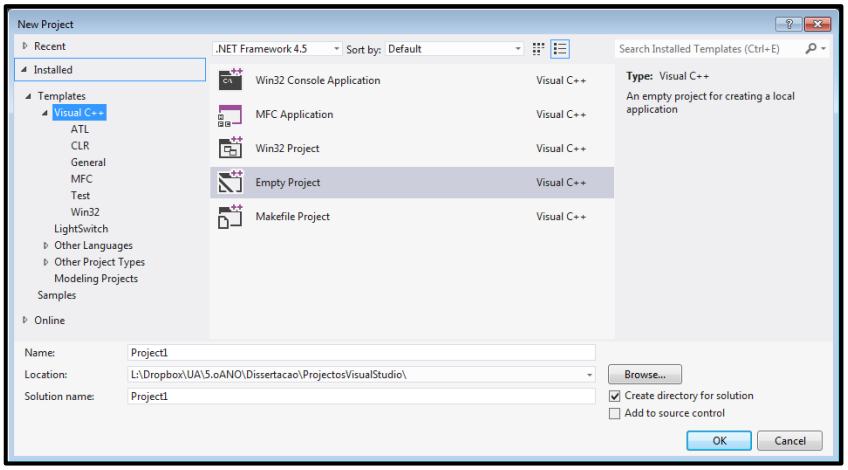

Figura 77 – Tipo de projeto

Quando o projeto C estiver concluído e pronto a testar deverá clicar-se em "Build" e de seguida escolher "Build <nome do projeto>", como apresentado na figura 78. Caso surjam erros ou avisos, essas informações poderão ser visualizadas na secção "Output", na área inferior da janela. Duplo clique nas entradas de erro ou aviso transportam a vista da janela principal para a linha onde esse erro ou aviso ocorreu. Caso a operação de "Build" tenha sido concluída com sucesso, o executável deverá estar presente na pasta do projeto.

Conclui-se assim a apresentação dos procedimentos de criação de aplicações de consola usando o Microsoft Visual Studio 2012.

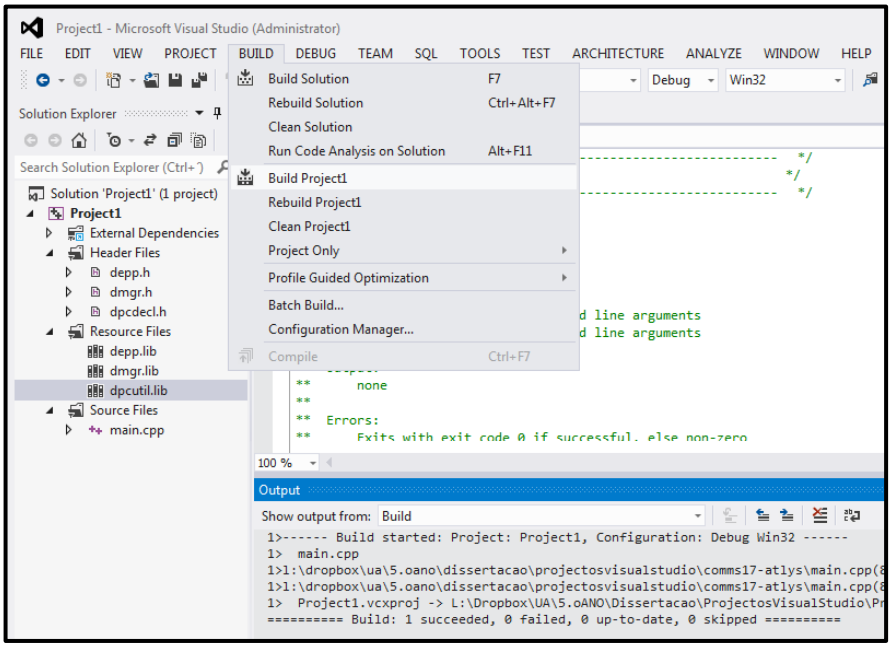

Figura 78 - *Build Project*

3.2.3 – Digilent Adept 2

Disponibilizada gratuitamente pela Digilent como suporte às suas placas com FPGA embutida, e compreendendo uma série de aplicações e utilitários agregados num GUI de fácil utilização, a aplicação Adept 2 agiliza uma série de operações úteis à programação da FPGA embutida na placa ou ao seu interface com os periféricos que estejam disponíveis [82].

A sua instalação transfere a aplicação GUI e também os drivers das ligações USB às diversas placas, permitindo o reconhecimento imediato das placas quando são ligadas a um PC.

Os procedimentos seguidos serão de seguida apresentados, na vertente Nexys 2 e Atlys, usando a iteração 2.10.2 desta aplicação.

O quadro inicial da aplicação Adept permite de imediato verificar no canto superior direito qual a placa selecionada para configuração, o refrescamento das ligações à placa através de "Initialize Chain" e uma área de configuração denominada "Config" que permite o carregamento de *BitStreams* diretamente na FPGA ou, caso disponível, na *Xilinx Platform Flash*. Em ambos os casos bastará clicar "Browse" para pesquisar o ficheiro de configuração, "OK" quando for encontrado e de seguida "Program" (vide figura 79).

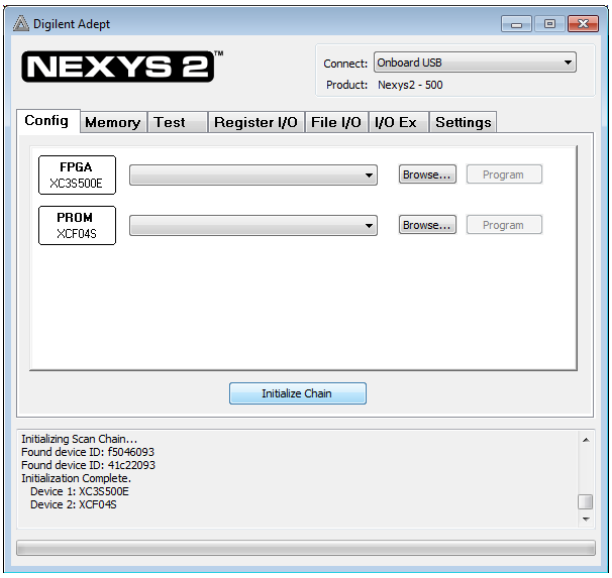

Figura 79 - Adept 2 - Área de programação das FPGA

Na área "Settings", apresentada na figura 80, é possível alterar o nome registado para as placas que estejam ligadas ao sistema. Para tal clique-se "Device Manager…"

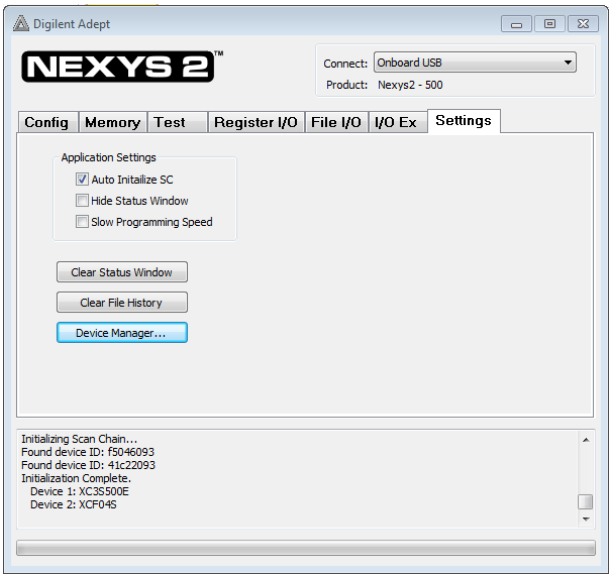

Figura 80 - Área de configurações da aplicação

Na caixa de diálogo aberta, apresentada na figura 81, é possível enumerar os dispositivos conectados. Clique-se em "Enumerate" e deverão surgir as várias placas ligadas. Escolha-se aquela à qual se pretende alterar o nome registado e introduza-se um novo nome na área "Alias". De seguida clique-se em "Add Dvc" e finalmente em "Save", ficando concluída a configuração do novo nome.

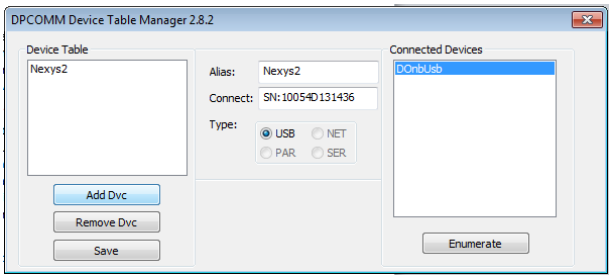

Figura 81 - Gestor de Dispositivos

O *tab* "Memory" permite o acesso operações sobre as memórias externas ao FPGA, permitindo escrever ou ler a partir de um ponto da memória ou simplesmente apagar os seus conteúdos, como apresentado na figura 82.

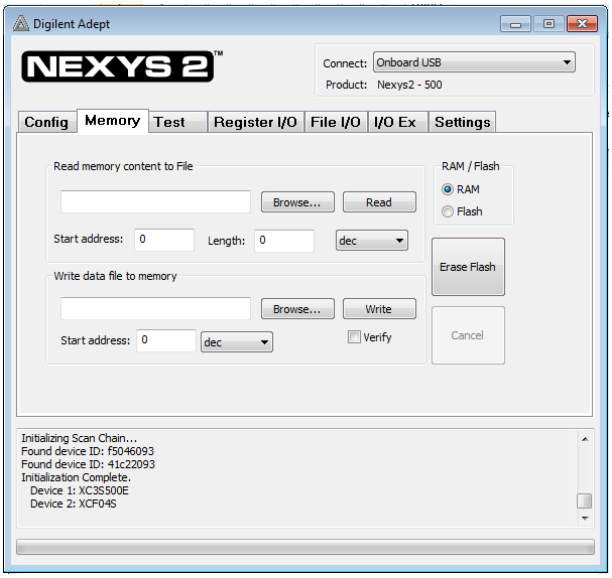

Figura 82 - Área de operações sobre as memórias externas

Paralelamente à aplicação e drivers disponibilizados como Adept 2 é também disponibilizado um SDK com documentação, bibliotecas e exemplos de código fonte para implementação na lógica da FPGA e código fonte destinado à criação de aplicações que usem a *Aplication Programming Interface* (API) desta aplicação. Estas possibilitam a execução de diversas tarefas que usam os interfaces e a FPGA, entre as quais uma implementação de UART, controlo de motores DC ou um interface de porta paralela assíncrona, entre outros [83].

3.2.3.1 – Digilent Plugin For Xilinx Tools – Utilização e Configuração

Este *plugin* permite a uma série de aplicações do Xilinx Design Tools utilizar diretamente os circuitos USB utilizados pelas placas Digilent para programar o FPGA como se

de um cabo JTAG se tratasse. Entre as aplicações que beneficiam deste *plugin* contam-se Xilinx iMPACT, Chipscope Pro e EDK. O pack que o inclui poderá ser descarregado no site da Digilent [84].

No arquivo poderão ser encontradas pastas adequadas às várias versões da Xilinx *Design Tools*. Dentro da pasta adequada à versão instalada pode ser encontrado um documento explicando com mais detalhe a instalação deste *plugin* e uma pasta "plugin" cujo conteúdo são duas pastas, "nt" e "nt64". Em função da versão de Windows instalada no PC (32 ou 64 bits) escolha-se a pasta "nt" ou "nt64" respetivamente e copie-se a pasta para a localização:

"X:\...\Xilinx\1X.X\ISE_DS\ISE\lib\"

optando pela fusão de pastas caso requerido. Caso esta operação tenha sido bem-sucedida, deverá ser possível encontrar os ficheiros "libCseDigilent.xml" e "libCseDigilent.dll" no caminho:

"X:\...\Xilinx\1X.X\ISE_DS\ISE\lib\nt\plugins\Digilent\libCseDigilent".

A configuração só ficará terminada identificando a placa usando a aplicação iMPACT.

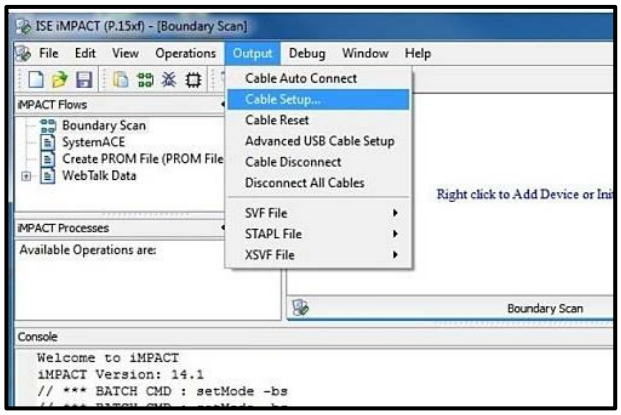

Figura 83 - iMPACT - GUI da aplicação e configuração do cabo 1

Inicie-se a aplicação e escolha-se "Cable Setup" na *tab* "Output" como executado na figura 83.

Caso a configuração do *plugin* tenha sido bem-sucedida deverá estar presente a opção "Digilent USB JTAG Cable" na caixa de diálogo apresentada para configuração do cabo de ligação. Escolha-se essa opção e clique-se em "OK", como realizado na figura 84.

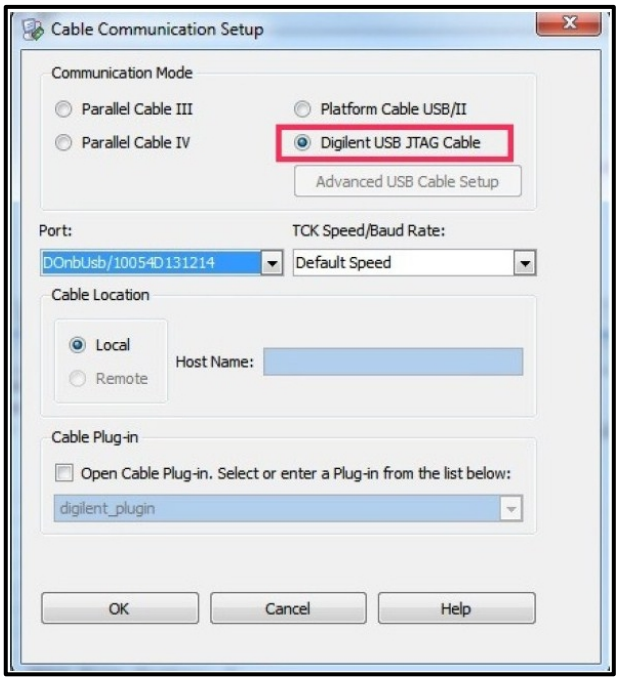

Figura 84 - Configuração do cabo 2

Clique-se na combinação "CTRL+I" para realizar a operação "Initialize Chain"

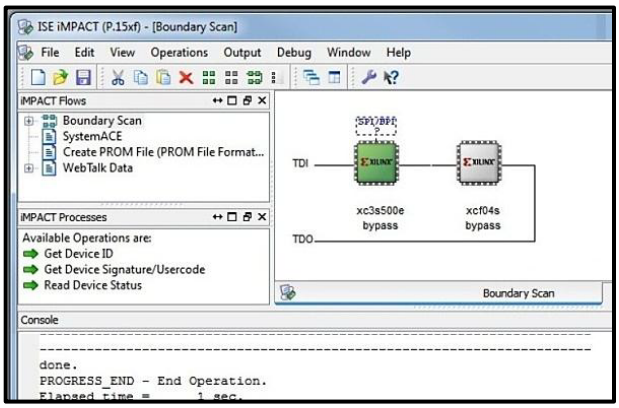

Figura 85 - Configuração do cabo 3

Deverá ser visível, como na figura 85, o *chip* presente na placa sendo ainda possível, no caso da Nexys 2, identificar a *Xilinx Platform Flash* "xcf04s". Está concluída a configuração do *plugin*, estando as placas disponíveis para configuração direta sem necessidade de utilização do pacote Adept 2.

3.2.4 – Notepad++

Esta aplicação simples, de utilização gratuita, é um editor de texto com funcionalidades avançadas e representa uma boa base de trabalho para tarefas de programação simples. De facto este editor de texto reconhece uma série de formatos

guardados em ASCII, como sejam os formatos C/C++ (extensões "C", "H" ou "CPP"), formatos de linguagem descritiva de *hardware* (extensões "VHD" ou "V"), formatos Matlab ("M") ou html("HTM"), não só apresentando esquemas de cores que permitem distinguir estruturas diferenciadas (por exemplo atribuindo cores a linhas '#define …' em C ou linhas de comentários '-- …' em VHDL) como também facilitando a identificação de ciclos ou outras estruturas fechadas por chavetas. Esta aplicação suporta a introdução de extensões adicionais através de *plugins*, ampliando assim as possibilidades desta aplicação em abrir e editar os mais diversos ficheiros. Possui modos de pesquisa e substituição refinados, possibilidade de inicializar outras aplicações que executem o ficheiro em edição, criação de macros e troca da codificação utilizada [85]. O GUI é apresentado na figura 86.

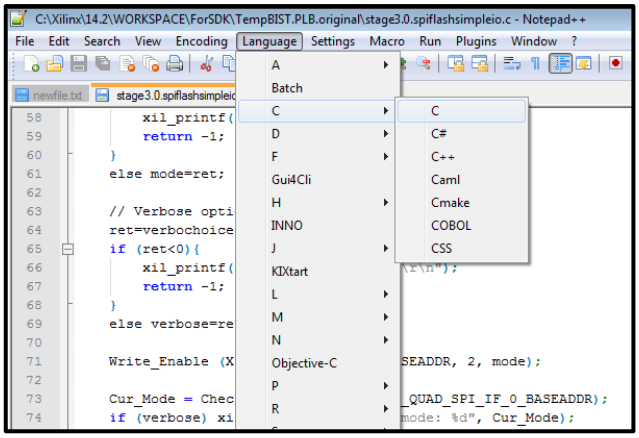

Figura 86 - Notepad++ - GUI da aplicação

3.2.4.1 – HEX-Editor Plugin for Notepad++

Um dos plugins que consideramos mais útil chama-se HEX-Editor Plugin for Notepad++ e adiciona ao Notepad++ a possibilidade de visualizar ficheiros no formato hexadecimal. Para além desta função primordial, realiza conversões entre hexadecimal e ASCII entre outras funcionalidades [86]. A sua apresentação é realizada através da figura 87.

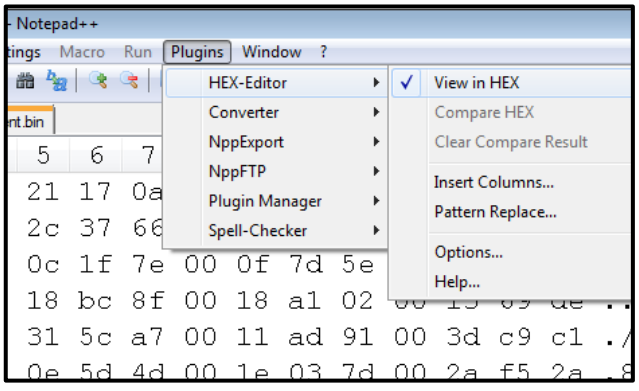

Figura 87 - Notepad++ - HEX-Editor Plugin

3.2.5 – Frhed

Alternativamente ao Notepad++ com o HEX-Editor Plugin e no sentido de visualizar rapidamente um maior volume de dados no formato hexadecimal poderá ser usada a aplicação Frhed – "Free hex editor" [87]. Esta aplicação fornece ainda uma função comparadora entre ficheiros com dados no formato hexadecimal, identificando rapidamente os índices das áreas com discrepâncias como ilustrado na figura 88.

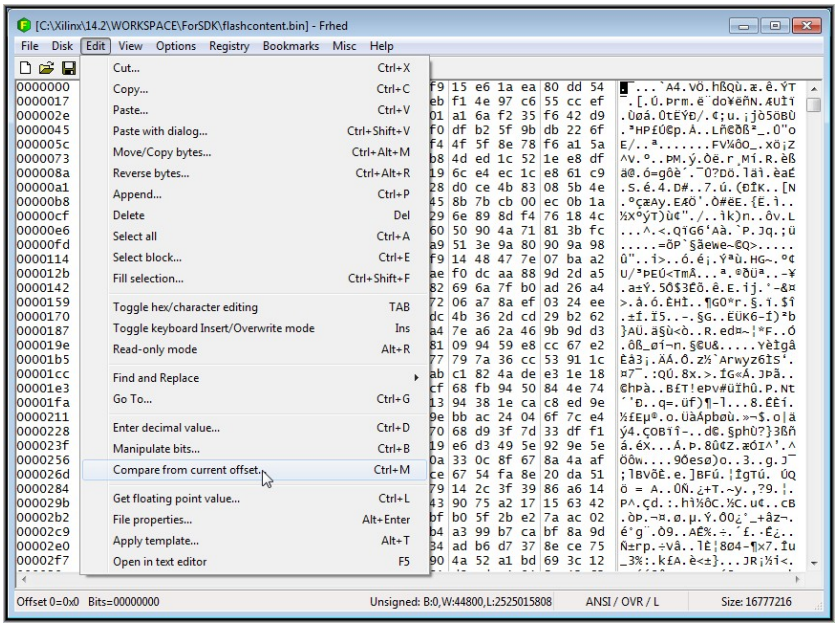

Figura 88 - Frhed - GUI da aplicação

3.3 – Conclusão

Neste Capítulo introduziram-se as diversas ferramentas utilizadas para a criação dos projetos e as configurações necessárias:

- *Hardware* Digilent Nexys 2 – 1200 e Atlys;
- Suite Xilinx para implementar *hardware* e soluções embutidas MicroBlaze;
- Visual Studio 2012 para a criação de aplicações de consola;
- O GUI Adept 2 que proporciona ao utilizador uma API de comunicação com as placas Digilent que o suportem;
- Notepad++ e Frhed para leitura e edição de ficheiros em formato binário.

No próximo capítulo apresentaremos os projetos propostos nos Objetivos.

Capítulo 4 – Projetos Desenvolvidos

Neste capítulo são apresentados os diversos projetos desenvolvidos destinados ao teste e transmissão de dados entre PC e placas Digilent.

Começa por abordar três ferramentas: geração, conversão e teste de dados pseudo-aleatórios. Passa a introduzir uma ferramenta de criação de ficheiros de coeficientes que podem ser carregados nas memórias internas das FPGA Xilinx no momento da programação. De seguida introduz um projeto de comunicação através do protocolo EPP que funciona sobre USB para transmissão de dados entre PC e FPGA. Finalmente apresenta-se um conjunto de projetos que visam o carregamento das memórias internas e externas das placas Digilent usando esse mesmo protocolo, controlado por um microprocessador MicroBlaze.

4.1 – Ferramentas de suporte a aplicações em FPGA

Dada a necessidade de testar os comportamentos dos diversos projetos de comunicação entre um PC e uma das placas Digilent em análise neste estudo, realizou-se um enquadramento deste trabalho com uma aplicação que muito beneficia do paralelismo dos FPGA: a ordenação de dados [88]. No entanto, diversos algoritmos e implementações em FPGA poderão beneficiar da utilização das seguintes ferramentas:

- Um gerador de dados de teste que opere num PC, criando ficheiros em formatos adequados à leitura pelo utilizador e ao tratamento pelo *hardware* presente na placa baseada em FPGA; entre outros, poderá ser necessária a criação de ficheiros binários adequados à escrita em memória *Flash*;
- Um projeto que traduza os dados de teste recebidos num PC após realização de operações na FPGA que envolvam o armazenamento em formato binário numa memória *Flash*;
- Um elemento que realize operações sobre os dados recebidos e que permita a sua validação, no caso presente realizando comparação entre as listas de dados enviados (na sua versão ordenada pelo CPU do Computador) e a lista de dados recebidos (ordenados pela implementação em FPGA).

As próximas secções abordarão as aplicações de consola criadas para concretizar estas ferramentas.

4.1.1 - Geração de Dados

Para a realização desta aplicação em C consideram-se as características do *hardware* em análise como também as necessidades típicas do utilizador no momento de criação do ficheiro ou ficheiros de dados de teste. Os procedimentos são os seguintes (figura 89):

- O executável deve ser invocado com dois parâmetros, número de valores de 32 Bit a gerar e nome comum para os ficheiros a criar;
- A aplicação inicializa criando as variáveis necessárias;
- É apresentada informação textual acerca da forma como devem ser introduzidos os parâmetros, os nomes dos ficheiros que serão gerados e a mensagem de conclusão da aplicação. Caso existam problemas com os parâmetros introduzidos é este conjunto de informações que é disponibilizado ao utilizador.
- É verificado o número de valores pedidos através de parâmetro à aplicação: caso seja ultrapassada a dimensão das memórias *Flash* presentes em ambas as placas (de 128 Mbit) a aplicação informa o utilizador, dando a hipótese de terminar a aplicação ou de prosseguir;
- Os diversos ficheiros serão criados pela aplicação e são abertos no modo de escrita:
    - Um ficheiro com os dados em formato ASCII hexadecimal;
    - Um ficheiro com os dados em formato ASCII decimal;

  - Um ficheiro com os dados em formato ASCII hexadecimal ordenados (usando o CPU do Computador);
  - Um ficheiro com os dados em formato ASCII decimal ordenados (usando o CPU do Computador);
  - Um ficheiro com os dados em formato binário com a dimensão das memórias *Flash* presentes na Nexys 2 e Atlys (aberto em modo de escrita binária);
- É alocada a memória necessária ao armazenamento dos valores pseudo-aleatórios.
- É iniciado um ciclo de geração e armazenamento dos valores pseudo-aleatórios;
- É impresso no ficheiro binário, destinado às memórias *Flash*, a quantidade de valores que a lista terá;
- Os valores são impressos nos diversos ficheiros;
- Caso o número de valores impressos no ficheiro binário não atinja a dimensão das memórias *Flash*, é realizado o seu preenchimento com o valor 0xFF (todas as células com valor '1');
- É realizada uma operação de ordenação para ordenação dos dados armazenados;
- São impressos os ficheiros de dados ordenados;
- São concluídos os procedimentos através da libertação da memória alocada e do fecho dos ficheiros.

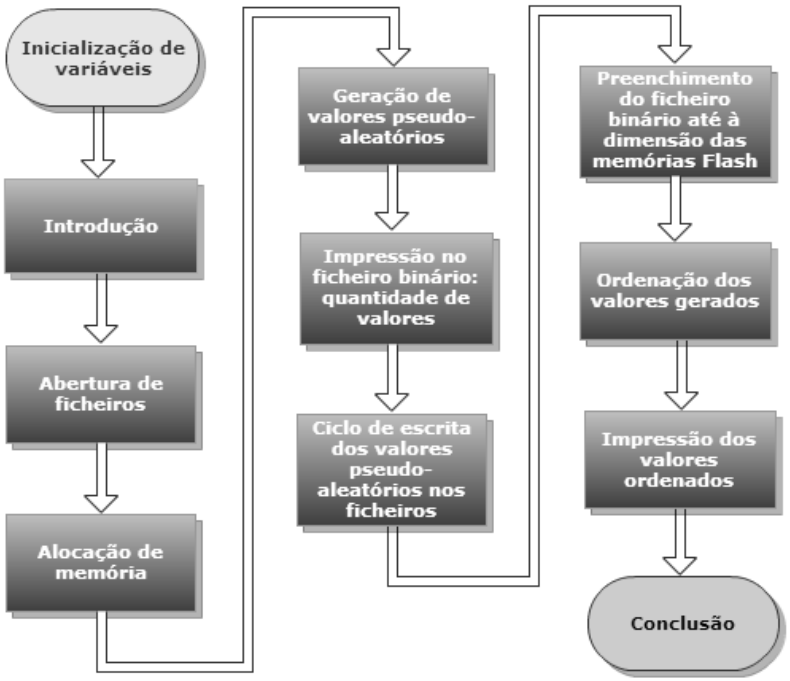

Figura 89 - Fluxo de projeto da aplicação geradora de dados pseudo-aleatórios

A primeira implementação testada para a geração de dados pseudo-aleatórios recorria à repetição da chamada da função rand( ) da biblioteca "stdlib.h" da linguagem ANSI-C.

No caso do Visual Studio 2012, o valor máximo que esta função pode atingir é de 0x7FFF, ou seja valores com até 15 bits, ficando aquém das necessidades de geração de dados

de 32 bits. Para colmatar esta situação procedeu-se à chamada da função em três iterações, obtendo por deslocamento dos bits de cada chamada os 32 bits de dados. Esta solução não é uniforme pelo que foi abandonada.

Apesar de não ser o foco desta Dissertação o cuidado com o tipo de dados aleatórios gerados, procedeu-se à introdução de um método validado, bem documentado e aceite como eficiente para a geração de palavras, no caso o Pacote Gerador de Números Pseudo-aleatórios (PRNG) *Mersenne Twister* [89], usando para tal código disponibilizado livremente *online* [90].

Quanto ao algoritmo de ordenação de dados, para impressão num ficheiro, optou-se inicialmente pela utilização de um algoritmo *Bubble Sort* pela simplicidade da implementação. No entanto, nos casos de teste com um elevado volume de dados, a operação de ordenação representava a grande percentagem do tempo demorado a concluir todos os passos do projeto. Para obviar esta situação introduziu-se igualmente um algoritmo válido, bem documentado e aceite como eficiente, no caso a ordenação *QuickSort* [91] usando igualmente código disponibilizado livremente *online* [92].

O código que concretiza este projeto pode ser encontrado no anexo A.

4.1.2 – Conversão de Dados

Após a transmissão de dados para a placa Digilent os algoritmos implementados leem os dados e alteram-nos de forma conducente aos resultados desejados. Poderá ser necessário recuperar os dados sobre os quais a implementação operou e proceder à sua conversão para um formato de fácil manuseio e leitura.

Em particular, os dados guardados nas memórias *Flash* de ambas as placas são regularmente salvos no formato binário, formato de difícil leitura em leitores de texto comuns tipicamente orientados a dados ASCII. Apesar das aplicações *freeware* apresentadas no capítulo 3 que permitem a visualização rápida dos dados no formato binário (na verdade em formato hexadecimal), os formatos visualizáveis num leitor de texto convencional são vantajosos pela sua compatibilidade com a maioria das aplicações de tratamento de dados e pelo facto de permitirem uma comparação visual imediata.

Refira-se ainda que o carregamento correto das memórias *Flash* implica a criação, escrita e leitura de ficheiros binários com a dimensão do espaço de armazenamento, independentemente do preenchimento efetivo de todo o ficheiro com dados. Desta forma, o carregamento do ficheiro binário para leitura implica o carregamento de 128 Mbits prejudicando o desempenho dos leitores de texto comuns e mesmo daqueles orientados ao formato binário.

Para obviar a complexidade de tratamento de ficheiros de grande dimensão em formato binário desenvolveu-se uma aplicação de conversão dos ficheiros recebidos para formatos texto “TXT”, que segue os processos descritos no diagrama de fluxo de sinal apresentado.

Os procedimentos descrevem-se de seguida (vide figura 90):

- São inicializadas as diversas variáveis necessárias;

- Procede-se à introdução, através da impressão de linhas de texto, do funcionamento do executável e dos argumentos requeridos ao seu funcionamento correto;
- Segue-se neste ponto a validação e configuração do número de valores a converter por leitura e tratamento do parâmetro numérico invocado através da linha de comandos;
- Realiza-se a leitura do argumento alfanumérico invocado através da linha de comandos que indica o nome do ficheiro sobre o qual realizar a conversão;
- Concluída esta operação é realizada a abertura desse ficheiro para leitura em formato binário e a abertura de dois outros ficheiros de texto destino em modo de escrita, um com dados em formato decimal e o segundo no formato hexadecimal;
- É executada a conversão cíclica dos dados lidos no ficheiro origem e a sua impressão no ficheiro destino;
- Os procedimentos concluem fechando os ficheiros abertos para leitura e edição e terminando o executável.

Figura 90 - Fluxo de projeto da aplicação conversora de ficheiros binários

Concluídas as operações o utilizador deverá ter disponível um ficheiro com os dados presentes no ficheiro binário convertidos para o formato ASCII decimal e outro com esses dados no formato ASCII hexadecimal.

O código associado a este projeto pode ser consultado no anexo B.

4.1.3 – Comparação de Dados

A validação das operações de ordenação ou outro tipo de simulação e teste pode ser conseguida contrastando os dados gerados para envio e os dados recebidos e previamente convertidos para um formato de mais fácil leitura. Este projeto compara os dados gerados através da aplicação geradora de dados (na forma do ficheiro com valores ordenados) com os dados recebidos (e possivelmente convertidos com o projeto apresentado na secção anterior).

Para o realizar é seguido o diagrama de fluxo de dados presente na figura 91. O executável percorre as seguintes etapas:

- É realizada a inicialização de variáveis e introdução textual à operação;
- A aplicação lê os nomes de ficheiros a comparar, invocados como parâmetros através da linha de comandos, e procede à sua abertura para leitura em formato de texto;
- Segue-se um ciclo de leitura, que lê ambos os ficheiros a comparar linha a linha e compara os dados presentes em ambas as linhas;
    - Caso seja encontrada uma diferença, é impresso no monitor a linha onde essa diferença pode ser encontrada e os valores identificados em ambos os ficheiros;
- A aplicação conclui apresentando um somatório de erros ou alternativamente indicando que os ficheiros são iguais;
- É realizado o fecho dos ficheiros abertos e terminado o executável.

O código associado a este projeto pode ser lido no anexo C.

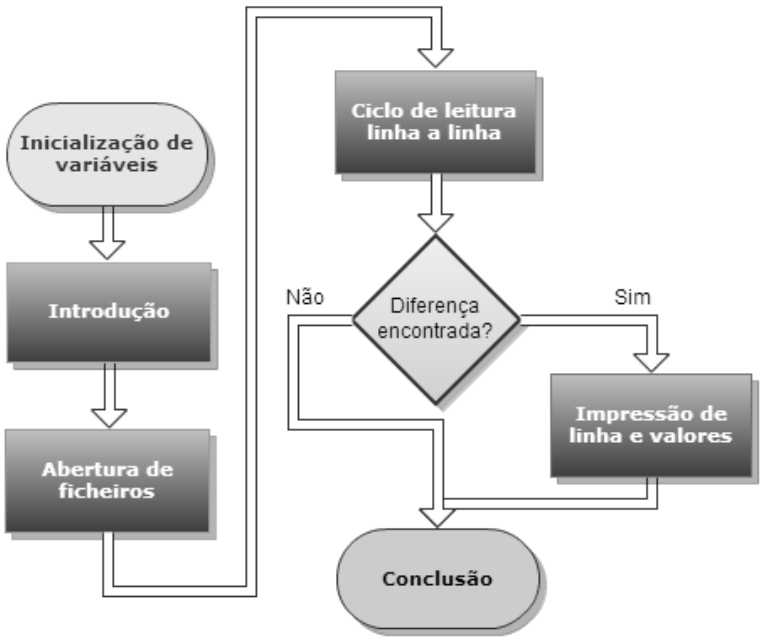

Figura 91 - Fluxo de projeto da aplicação comparadora de dados

4.2 – Gerador de ficheiros "COE" para carregamento de dados

Uma das formas de disponibilizar aos algoritmos implementados na *fabric* da FPGA valores de teste aleatórios é o carregamento direto nos blocos RAM embutidos, através de ficheiros de coeficientes "COE". Utilizáveis no momento da geração de blocos RAM ou filtros, entre outros blocos que aceitam este tipo de ficheiro, através da funcionalidade "Core Generator" presente na aplicação ISE, são ficheiros com os dados de inicialização dos blocos mencionados. Estes dados que serão carregados nos blocos no momento da programação da FPGA dependem de alguns procedimentos para uma correta leitura [93]:

- A identificação da base dos valores, inscrevendo no ficheiro uma palavra apropriada seguida da base utilizada para escrita dos dados;
- A identificação do tipo de bloco que utilizará o ficheiro "COE", imprimindo a palavra apropriada seguida dos valores a carregar;
- Os diversos valores numéricos que constituem o vetor a carregar deverão ser separados entre si por uma vírgula ",";
- Qualquer caracter ou carateres precedidos do ponto e vírgula ";" são interpretados como comentários.

Conhecidas estas regras de criação do ficheiro concretizem-se as configurações possíveis na forma dos diversos códigos possíveis, apreciáveis nas tabelas 4 e 5.

Sintaxe de ficheiros "COE" – primeira palavra de configuração		
Palavras base	RADIX	radix = 10;
	MEMORY_INITIALIZATION_RADIX	memory_initialization_radix = 2;

Tabela 4 - Sintaxe de ficheiros COE: primeira palavra

São apresentadas duas palavras raiz associadas a dois grupos de blocos alvo: a palavra "RADIX" é destinada a ficheiros "COE" que irão carregar coeficientes; a palavra "MEMORY_INITIALIZATION_RADIX" é destinada a inicializar blocos de memória.

Sintaxe de ficheiros "COE" - segunda palavra de configuração		
Palavras de dados	COEFDATA	coefdata = a, 8, 2, f, d, 0,…
	MEMORY_INITIALIZATION_VETOR	memory_initialization_vetor = 02, f6, 84,…
	PATTERN	pattern = 1, 2, 3, 1, 2, 3,…
	BRANCH_LENGTH_VETOR	branch_length_vetor = 3, 10, 20, 40,…
	MEMDATA	memdata = 0, 1, 0, 1, 1,…

Tabela 5 - Sintaxe de ficheiros COE: segunda palavra

São apresentadas as cinco palavras de dados associadas aos vários tipos de blocos que aceitam inicialização através de ficheiros "COE":

- A palavra "COEFDATA" significa que os dados que se seguem são coeficientes;
- A palavra "MEMORY_INITIALIZATION_VETOR" indica que os dados que se seguem são dados de inicialização de blocos de memória;

- A palavra “PATTERN” indica dados que se destinam à inicialização de blocos de correlação binários;
- A palavra “BRANCH_LENGTH_VETOR” precede dados destinados a atuar como coeficientes dum bloco de entrelaçamento/desentrelaçamento;
- Finalmente a palavra “MEMDATA” é uma palavra obsoleta associada a blocos de memória mas é mantida neste projeto para garantir compatibilidade com implementações mais antigas.

Apresenta-se na figura 92 o diagrama de fluxo de sinal seguido para a construção da aplicação geradora de ficheiros “COE”.

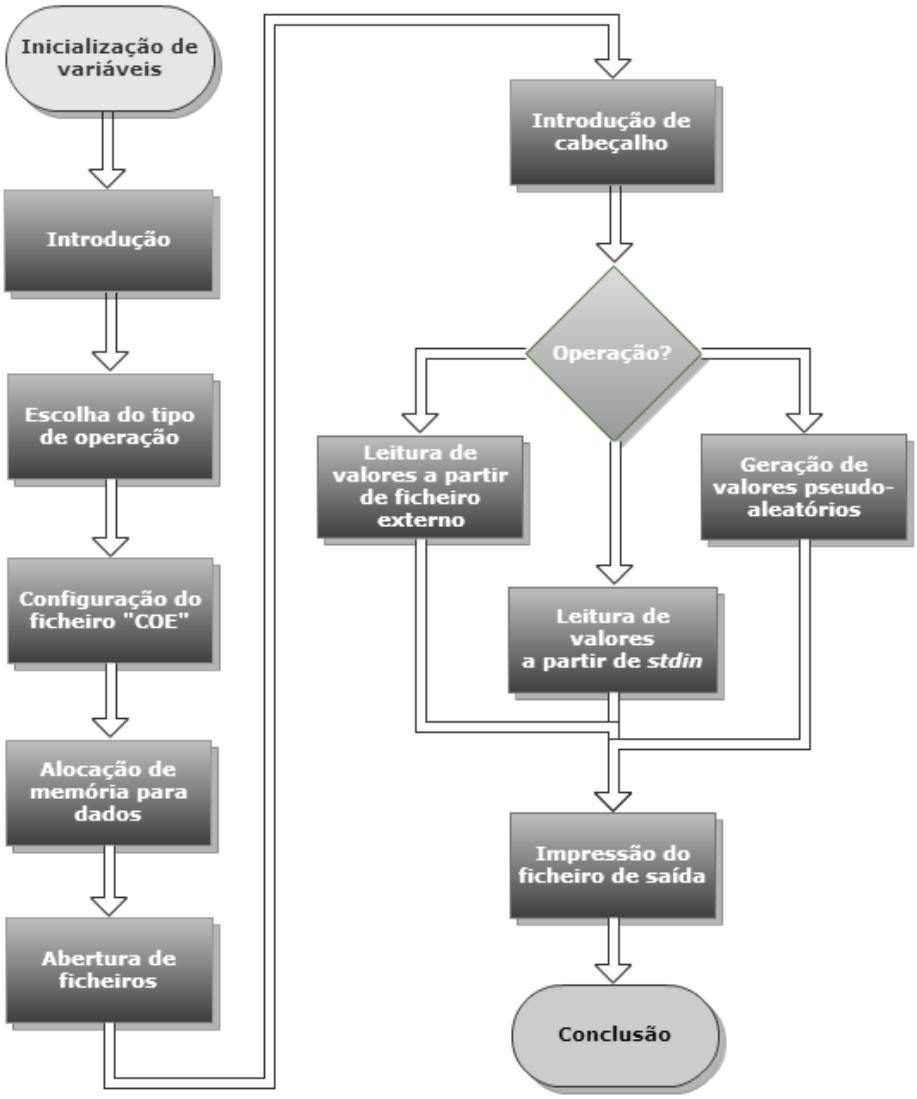

Figura 92 - Fluxo de projeto da aplicação geradora de ficheiros COE

O processo de criação do ficheiro “COE” passa pelas seguintes etapas:

- Uma fase inicial compreende a inicialização de variáveis necessárias à aplicação;

- Segue-se a impressão da introdução que esclarece as funcionalidades do projeto e as opções requeridas para a geração do ficheiro;
- A escolha do tipo de operação pretendida é a etapa seguinte:
    - A conversão para ficheiro "COE" de dados presentes num ficheiro externo, que deverá ter sido fornecido como argumento ao executável;
    - O preenchimento do ficheiro com dados pseudo-aleatórios, usando o PRNG *Mersenne Twister* já apresentado na secção anterior;
    - O preenchimento do ficheiro com dados introduzidos através de *stdin*.
- Escolhida a operação a seguir mais tarde na execução, configura-se o ficheiro "COE" incluindo a quantidade de valores a escrever e a dimensão desses valores (no caso binário e hexadecimal, o número de Bit pretendido), a raiz a utilizar (binária, decimal ou hexadecimal) e finalmente o tipo de dados (de inicialização de memória, coeficientes entre outros).
- A execução continua alocando o espaço de memória necessário às operações de geração de dados;
- Procede-se com a abertura dos ficheiros para escrita onde dos dados serão guardados;
- Em função da configuração da base e do tipo de dados é possível identificar o cabeçalho a imprimir nas primeiras linhas do ficheiro;
- Realiza-se a operações de leitura ou geração de dados de acordo com a escolha realizada a montante;
- Concluída a operação os dados são impressos no ficheiro;
- As operações são concluídas com a limpeza das variáveis alocadas.

Fica concluída assim a criação do ficheiro "COE".
O código criado para este projeto pode ser encontrado no anexo D.

4.3 – Transferência de dados através da interface USB-EPP

A aplicação Adept 2 é a ferramenta já introduzida no Capítulo 3 que permite ao utilizador o carregamento de ficheiros de programação da FPGA "BIT", escrita e leitura da memória *Flash* externa, teste de periféricos entre outras operações. A sua instalação armazena em disco os *drivers* indispensáveis ao funcionamento da porta USB, permitindo o reconhecimento das placas compatíveis com esta aplicação assim que são ligadas a um Computador de uso geral. Para além do *driver* são instaladas bibliotecas de ligação dinâmica (DLL) que permitem a realização das operações mencionadas através de USB, usando o GUI da própria aplicação ou usando funções disponibilizadas para implementação em novos projetos no SDK disponibilizado pela Digilent [94].

Um dos projetos que é possível implementar na forma de uma aplicação de consola é o envio e receção de dados na forma de valores numéricos. Presente no SDK da aplicação Adept2 pode encontrar-se o código em VHDL que permite implementar as funcionalidades de escrita e leitura de dados nos registos da FPGA e o código com as funções em C que permitem o envio e receção desses dados através de USB. Este procedimento usa o protocolo *Enhanced Parallel Port* (EPP) que o *firmware* instalado no *chipset* que controla a porta USB emula. O código VHDL está presente no ficheiro "dpimref.vhd" e o código com as funções C que permitem a comunicação está presente no ficheiro "DeppDemo.cpp".

Como é possível analisar no documento introdutório a este mecanismo de comunicação, também presente no SDK sob o nome "Digilent Asynchronous Parallel Interface (DEPP)", o interface EPP é implementado mediante três tipos de operações de transferência de dados: escrita de endereços, escrita e leitura de dados.

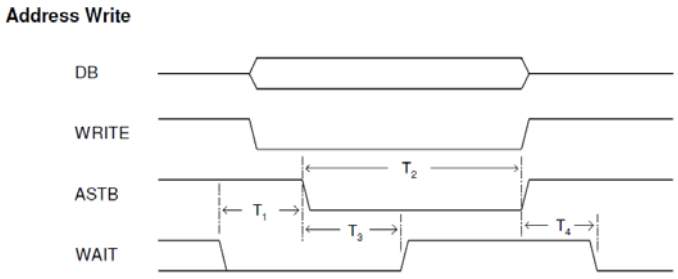

Figura 93 - Sinalização da operação de escrita de endereço [95]

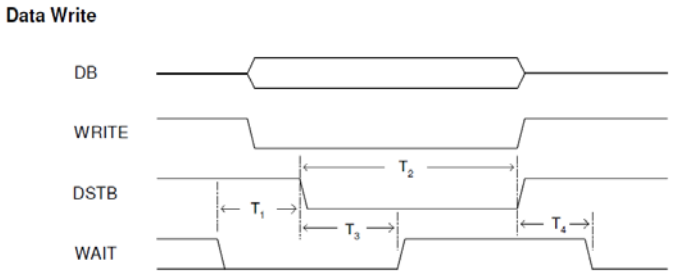

Figura 94 - Sinalização da operação de escrita de dados [96]

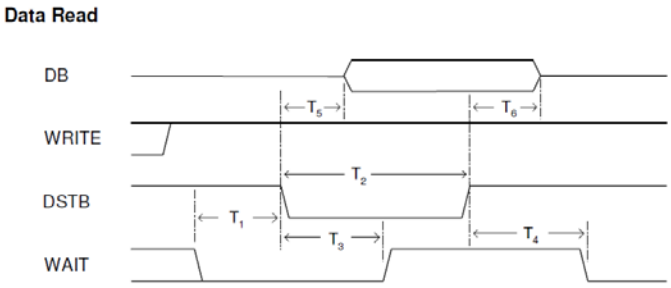

Figura 95 - Sinalização da operação de leitura de dados [97]

As figuras 93, 94 e 95 ilustram a sinalização realizada para a execução de cada uma destas operações. Os sinais relevantes são:

- Sinal WRITE, significando "escrita", é um sinal de entrada na lógica da FPGA; toma valor '0' para operações de escrita e valor '1' para operações de leitura, indicando desta forma a direção da transferência;
- Sinal ASTB, significando "*strobe* do endereço", é um sinal de entrada na lógica da FPGA; toma valor '0' para operações de escrita do endereço;
- Sinal DSTB, significando "*strobe* dos dados", é um sinal de entrada na lógica da FPGA; toma valor '0' para operações de escrita ou leitura de dados;
- Sinal WAIT, significando "aguardar", é um sinal de saída da lógica da FPGA; toma valor '0' quando o periférico está pronto a receber nova transferência de dados e valor '1' quando a operação de escrita ou leitura é concluída na FPGA, estimulando a conclusão da operação no Computador de uso geral.

Repare-se que os sinais ASTB e DSTB nunca podem tomar simultaneamente valor '0' durante uma das operações possíveis. Note-se ainda que o sinal WAIT, que como mencionado realiza duas sinalizações através dos dois estados possíveis, deverá tomar valor '1' no início da transferência ou tomar valor '0' na conclusão durante uma janela de tempo de 10 ms, caso contrário é sinalizado pela aplicação que opera no PC anfitrião um erro de *timeout* (intervalos T3 e T4 presentes nas figuras 93 a 95).

A análise do código VHDL presente no ficheiro "dpimref.vhd" exibe discrepâncias entre a estrutura a implementar na FPGA e o tipo de operações que este interface pode realizar: enquanto o documento aponta para três operações, o código VHDL implementa quatro operações (sendo a operação de leitura de endereço esse quarto elemento). A análise de documentação de versões desatualizadas do SDK e de documentação descontinuada da Digilent demonstra que existe um atraso do código em relação à solução apresentada de forma documental.

Associado a este interface que deverá ser instanciado na FPGA, são utilizadas funções fornecidas pela Digilent e que iniciam a transferência em ambos os sentidos, realizando pedidos de escrita e de leitura de conteúdos através de USB usando o PC anfitrião.

De facto a *Application Programming Interface* (API) do Adept 2 reconhece as seguintes funções orientadas ao envio de endereço e dados de dimensão Byte, implementadas no ficheiro "DeppDemo.cpp":

DoPutReg() – Função que toma como parâmetros o endereço e a palavra de dados que o utilizador pretenda enviar do anfitrião à FPGA e que realiza as operações necessárias ao envio de ambos até à lógica da FPGA;
DoGetReg() – Função que toma como parâmetro o endereço onde o utilizador pretenda ler dados, realizando as operações necessárias ao envio do endereço até à FPGA e à leitura dos dados que lhe estejam associados.

Esses parâmetros de acesso são *szRegister* e *szByte* para o endereço e o valor a enviar, respetivamente. Cada função realiza duas operações em sequência, não controláveis pelo utilizador: escrita de endereço seguida de leitura ou escrita de dados.

A implementação deste interface tal como é proposto apresenta limitações ao carregamento de dados nas placas Digilent por diversos motivos:

- O primeiro está relacionado com a dimensão das palavras que é possível transmitir, neste caso de dimensão 8 bits. Assim será possível o armazenamento de apenas 2^8 = 256 palavras cujo valor absoluto poderá variar entre 0 e 2^8-1 = 255. Caso se pretenda o carregamento de dados para efeitos de teste de algoritmos de ordenação, ambos os valores se apresentam muito reduzidos para permitir a sua validação.
- O segundo motivo relaciona-se com a máquina de estados sugerida pelo código VHDL fornecido. Esta apresenta a operação adicional de leitura de endereço e necessita que uma das etapas do fluxo de projeto para implementações em FPGA, a etapa de síntese, seja configurada de forma particular.
- O terceiro motivo prende-se com a clareza do código VHDL disponibilizado e a acessibilidade que este apresenta à introdução de alterações.

Para ultrapassar estas limitações e particularidades, e no sentido de projetar uma aplicação capaz de ler da linha de comandos ou de um ficheiro externo um volume superior de palavras de dimensão maior que 8 bits, optou-se pela substituição do código VHDL base fornecido por outro de utilização livre disponível online [98]. A versão utilizada é totalmente síncrona e de leitura comparativamente mais acessível, não requerendo a desativação das otimizações de codificação de máquinas de estados que a versão fornecida pela Digilent necessita. Esta base de trabalho apresenta no entanto as mesmas limitações em relação à dimensão do espaço de endereçamento e das palavras de dados.

Dado que as ferramentas desenvolvidas nas secções anteriores geram palavras de dimensão 32 bits foram introduzidas alterações no código VHDL e no código C que permitem o envio e leitura de palavras dessa dimensão. Sabendo que cada operação de escrita ou leitura de dados é precedida por uma operação de escrita de endereço, por cada 32 bits de palavra de dados podem também ser escritos num registo da FPGA 32 bits de palavra de endereço. Para armazenar um volume de dados considerável e de forma que não penalize a ocupação de malha lógica para o realizar, optou-se pelas memórias internas das FPGA presentes em cada placa Digilent. No caso da Spartan-3E presente na Nexys 2 são disponibilizados 504 Kbits e no caso da Spartan-6 presente na Atlys são disponibilizados 2088 Kbits. Se cada espaço for preenchido com elementos de 32 bits, serão necessários respetivamente 14 e 17 bits respetivamente para endereçar a totalidade dos blocos RAM. Por motivos de simplificação do

código produzido optou-se pela abdicação do armazenamento de 1.280 palavras das 66.816 possíveis numa Atlys, perdendo cerca de 2% da área de armazenamento, e limitou-se a dimensão dos endereços a 16 bits para ambas as placas. Os blocos RAM foram implementados usando o IP *Core* adequado da aplicação Xilinx ISE e foi adicionado ao projeto um IP *Core* de síntese de relógio para o controlo da frequência de operação dos diversos blocos da máquina de estados e da memória mencionada, usando procedimentos já descritos no capítulo 3 para a configuração de IP *Core*.

O procedimento de envio e receção passa pela utilização de iterações para escrita ou leitura de dados. Utilizando as funções C supracitadas e usando a decomposição e filtragem de palavras de dados e endereço, é possível a entrega byte a byte do endereço e da palavra de dados. As operações realizadas sobre ambas são descritas na tabela 6. Como pelo canal USB-EPP só podem ser transferidos 8 bits, para realizar a escrita de uma palavra de 32 bits realiza-se uma filtragem do Byte mais significativo de dados e do endereço alvo e procede-se ao seu envio; de seguida realiza-se uma filtragem do segundo Byte mais significativo de dados e do Byte menos significativo do endereço e realiza-se o seu envio para a FPGA. As duas iterações seguintes transferem os Bytes restantes da palavra de dados. A filtragem das palavras enviadas é possível graças à utilização do operador "&" ou "AND" e do operador ">>" ou "Shift Right". A lógica implementada na FPGA reconstrói a palavra recebida guardando nos bits adequados do registo de receção cada iteração de dados ou endereço.

Procedimento de escrita de uma palavra de 32 bits							
Iteração	**Palavra**		**Shift**	**Palavra**		**Shift**	**Bytes enviados**
	Endereço	**Filtro**		**Dados**	**Filtro**		
1	0x**05**BF	0x**FF**00	>>8	0x**A8**B7C6D5	0x**FF**000000	>>24	0x**05;**0x**A8**
2	0x05**BF**	0x00**FF**		0xA8**B7**C6D5	0x00**FF**0000	>>16	0x**BF;**0x**B7**
3	0xXXXX	0xXXXX		0xA8B7**C6**D5	0x0000**FF**00	>>8	0x**XX;**0x**C6**
4	0xXXXX	0xXXXX		0xA8B7C6**D5**	0x000000**FF**		0x**XX;**0x**D5**

Tabela 6 – Procedimento de escrita de uma palavra de 32 bits

Quanto ao procedimento de leitura de dados começa com o pedido da aplicação anfitriã que deve enviar o endereço por forma a recuperar o valor que lhe esteja associado. O procedimento é o descrito através da tabela 7 .

Procedimento de leitura de uma palavra de 32 bits							
Iteração	**Palavra**		**Shift**	**Byte**		**Shift**	**Palavra Dados**
	Endereço	**Filtro**		**Enviado**	**Recebido**		
1	0x**05**BF	0x**FF**00	>>8	0x**05**	0xXX		0xXXXXXXXX
2	0x05**BF**	0x00**FF**		0x**BF**	0x**A8**	<<24	0x**A8**XXXXXX
3	0xXXXX	0xXXXX		0xXX	0x**B7**	<<16	0xA8**B7**XXXX
4	0xXXXX	0xXXXX		0xXX	0x**C6**	<<8	0xA8B7**C6**XX
5	0xXXXX	0xXXXX		0xXX	0x**D5**		0xA8B7C6**D5**

Tabela 7 - Procedimento de leitura de uma palavra de 32 bits

A lógica implementada na FPGA recebe e reconstrói o endereço, usando-o para recuperar o dado da área de armazenamento numa sequência que requer duas iterações para

obtenção do endereço completo. Os dados são então recuperados em quatro iterações, uma por cada Byte de dados, que filtram e deslocam cada Byte até à posição adequada na palavra, sendo a primeira iteração de recuperação de dados comum à última iteração de escrita do endereço.

No sentido de agilizar a escrita ou leitura de dados em ficheiros foram desenvolvidas funções que abrem ficheiros para o efeito e implementam as várias iterações de envio ou receção de dados repetidamente para cada par endereço/dados.

Nesta fase testou-se o débito para ambos os tipos de operações, escrita e leitura de dados, usando funções disponibilizadas pela biblioteca "windows.h" para a contagem do tempo decorrido entre o início e fim das operações usando um elevado número de palavras de dados. Os testes retornaram um valor comum tanto para a placa Nexys 2 como para a Atlys operando à frequência do oscilador presente em cada uma (respetivamente de 50 e 100 MHz): cerca de 1,6 KB/s para operações de escrita e 1,3 KB/s para operações de leitura. Dado que as operações são síncronas testou-se a variação do sinal de relógio fornecido à máquina de estados para ambos os projetos, usufruindo do IP *Core* instanciado para o efeito, no sentido de aumentar o débito das transferências. No entanto os resultados não apresentaram alteração repetindo-se débitos similares aos já mencionados.

A implementação da interface realizada, comum para Nexys 2 e Atlys, está presente no anexo E. Um extrato do código crítico realizado para o envio de cada palavra de dados com 32 bits pode ser encontrado no anexo F. O código idêntico para operações de leitura de dados pode ser encontrado no anexo G. Os ficheiros UCF que realizam o mapeamento dos pinos da FPGA nas placas estão disponíveis na página da Digilent de cada uma das placas [99][100].

4.4 – Projetos baseados no microprocessador MicroBlaze

Nesta secção abordar-se-ão os diversos projetos usando o microprocessador *Soft Core* previamente estudado MicroBlaze. Será analisada para cada caso a ocupação de recursos da FPGA e os algoritmos desenvolvidos para estabelecer ligações entre PC e FPGA, ou seja a aplicação para consola desenvolvida em C que correrá no Computador e o *hardware* programado no IC para atendimento e execução dos comandos que dessa aplicação sejam iniciados. Primeiro serão introduzidos os procedimentos para envio e receção de dados seguindo-se a apresentação dos projetos implementados nas FPGA presentes na Nexys 2 e na Atlys.

4.4.1 – Transferência de dados genérica entre PC e FPGA

Utilizando um mecanismo USB-EPP idêntico ao previamente estudado na secção 4.3, este projeto baseado no Xilinx SDK está orientado para a receção e envio de palavras de 32 bits usando endereços de até 32 bits. As funções utilizadas permanecem *DoPutReg()* e *DoGetReg()*, da API da aplicação Adept 2, tendo por parâmetros *szByte* e *szRegister* para palavra de dados e de endereço respetivamente. Irá realizar-se de seguida a apresentação do processo de inicialização do código desenvolvido na aplicação Xilinx SDK no microprocessador MicroBlaze.

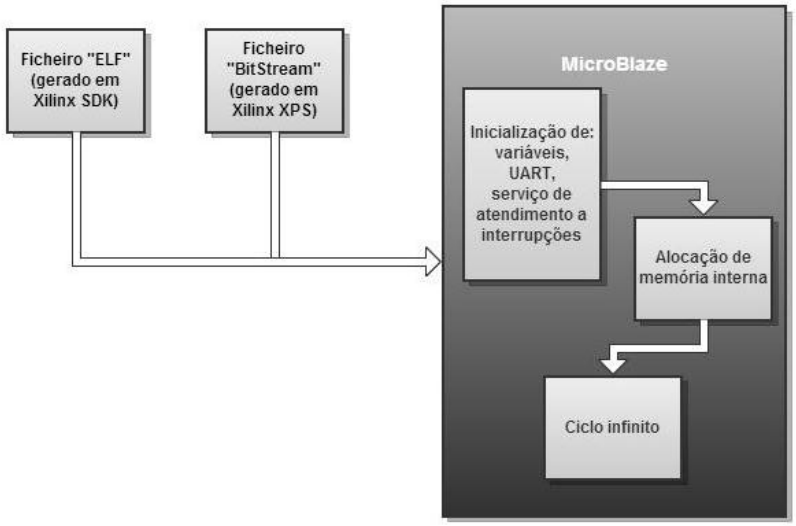

Figura 96 - Procedimento básico de operação das aplicações criadas usando Xilinx SDK

A figura 96 representa a seguinte sequência de operações:

- A aplicação Xilinx SDK programa o ficheiro "ELF", relativo ao código produzido na linguagem C para operação usando o microprocessador MicroBlaze, e o ficheiro "BitStream", relativo ao *hardware* a instanciar que foi criado na aplicação Xilinx XPS;
- Programada a FPGA, a aplicação em funcionamento no IC inicializa as variáveis necessárias ao seu funcionamento, configura e ativa a UART e configura, inicia e ativa

o serviço de atendimento a interrupções do MicroBlaze de forma a servir o protocolo USB-EPP;
- De seguida é alocada a área de memória interna pretendida para armazenamento de dados, caso seja necessária e esteja configurada essa operação;
- A aplicação entra então num ciclo *while* que permita que a aplicação não termine, ficando assim disponível a atender as interrupções originadas pelo protocolo USB-EPP.

A configuração do serviço de UART e do serviço de atendimento a interrupções é realizado usando funções das bibliotecas disponibilizadas para MicroBlaze para ambos os serviços. No caso dos diversos projetos apresentados, as bibliotecas usadas foram "xuartns550_l" e "xintc" respetivamente para UART e serviço de atendimento a interrupções, através da associação dos *headers* de cada biblioteca ao projeto em C. Quanto aos identificadores dos dispositivos instanciados, cujo conhecimento é essencial a esta configuração, podem ser obtidos no *header* "xparameters.h". Um exemplo de configuração de ambos os dispositivos está presente no anexo H, onde os identificadores mencionados começam pelo termo "XPAR".

O serviço de atendimento a interrupções funciona de forma sequencial: dado que as palavras a guardar são de 32 Bit, e cada ciclo de leitura ou escrita de dados através de USB-EPP só é capaz de transmitir 8 Bit, são necessários quatro ciclos para a totalidade da palavra estar disponível à FPGA e durante esse processo receber até 32 Bit de endereço.

Comece-se por conhecer a aplicação que inicia a interrupção, em funcionamento num PC. No caso da operação de escrita de um valor de 32 Bit num determinado endereço, a aplicação de consola em operação no Computador é invocada com três parâmetros:

- O identificador de tipo de operação pretendida (escrita) e o endereço alvo dessa operação;
- O identificador de nome do dispositivo e o nome configurado pelo utilizador ou por omissão para a placa alvo dessa operação;
- O identificador de valor a carregar seguido do mesmo.

Conhecidos e validados os valores introduzidos inicia-se o procedimento de envio de dados. A figura 97 apresenta o procedimento genérico de armazenamento de valores em registos da FPGA, blocos de RAM embutida ou nas memórias periféricas baseadas em SDRAM. Segue-se o procedimento associado ao serviço a interrupções que tem lugar na placa com FPGA:

- É recebido o primeiro par de Bytes de dados e endereço sendo ambos armazenados temporariamente; é realizado um deslocamento à esquerda da informação recebida, até ao Bit 31 (correspondendo aos bits mais significativos);
- É recebido o segundo Byte de dados e endereço e realizados novos deslocamentos à esquerda até ao Bit 23, sendo somados às respetivas variáveis temporárias;
- É recebido o terceiro Byte de dados e endereço e repetido o procedimento anterior até ao Bit 15;

- É recebido o quarto e último Byte de dados e endereço: os dados estão presentes na totalidade na FPGA e podem ser armazenados na memória alvo da operação, na posição indicada pelo endereço recebido.

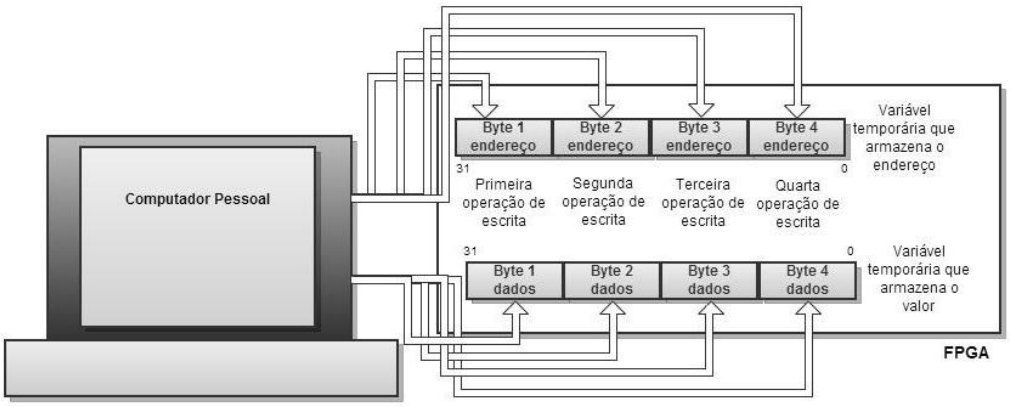

Figura 97 - Envio de 32 bits de endereço e 32 bits de dados para a FPGA

Para o procedimento de leitura as operações são similares. Para a execução da aplicação de consola, em operação num Computador, são fornecidos dois parâmetros à aplicação, a saber:

- O identificador de tipo de operação pretendida (leitura) e o endereço alvo dessa operação;
- O identificador de nome do dispositivo e o nome configurado pelo utilizador ou por omissão para a placa alvo dessa operação.

Inicia-se o procedimento de leitura de dados. Neste exemplo, ilustrado pela figura 98, os endereços e os dados serão de 32 Bit:

- A aplicação realiza as três operação iniciais de leitura enviando os três primeiros Bytes do endereço;
- Atingida a quarta operação estará disponível todo o endereço para utilização; é realizada a operação de leitura na memória alvo das aplicações;
- Na mesma iteração é retornado, através do canal USB-EPP, o Byte mais significativo da palavra de 32 Bit lida; esse valor é recebido na aplicação de consola que opera no PC, sendo armazenado numa variável temporária e deslocado até ao Bit 31.
- São repetidos os procedimentos de envio de dados, sendo os dados que correspondem à iteração da operação retornados ao PC; concluída a receção o valor é armazenado ou mantido numa variável temporária e impresso para leitura pelo utilizador.

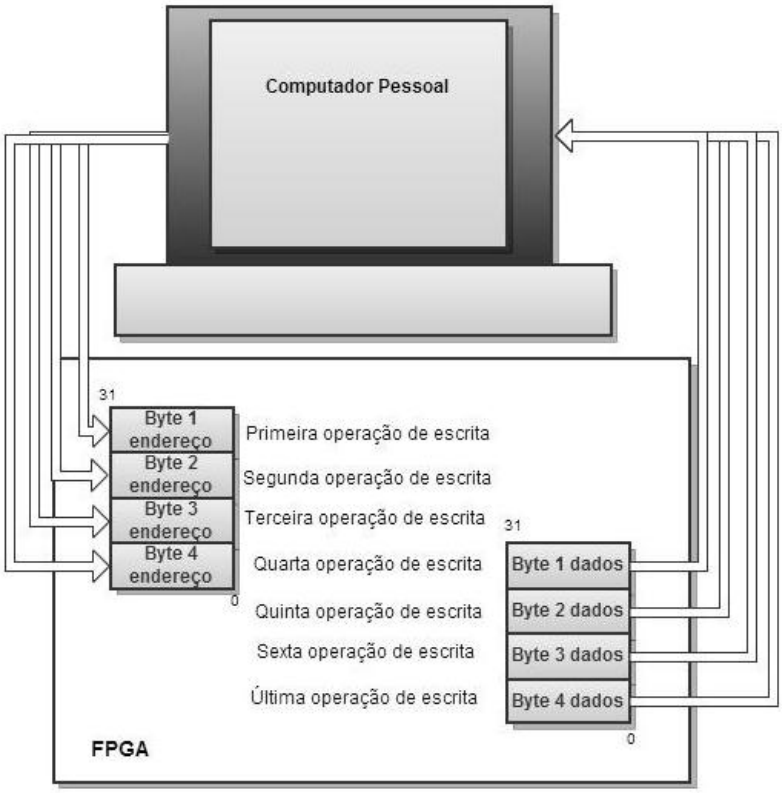

Figura 98 - Envio de 32 bits de endereço e receção de 32 bits de dados da FPGA

Este é o procedimento genérico de escrita e leitura em memórias voláteis, incluindo a memória interna da FPGA na forma de blocos RAM e as memórias SDRAM e DDR2-SDRAM presentes na Nexys 2 e Atlys respetivamente. Se no caso da memória interna é possível realizar armazenamento e leitura através da alocação de memória, usando para tal a função "malloc()" da biblioteca de ANSI C "stdlib", no caso das memórias externas voláteis é possível escrita e leitura usando funções da biblioteca "xio" disponibilizada para MicroBlaze: "xio_out32()" para escrita de valores de 32 bits e "xio_in32()" para leitura, por exemplo.

Estas funções são utilizadas igualmente para escrita e leitura das memórias não voláteis sendo no entanto incluídas em funções mais complexas. De facto a escrita e leitura destes periféricos obrigam à escrita e leitura de palavras de comando que inicializam as memórias *Flash* nos modos de leitura ou escrita. Nas secções onde serão apresentados os projetos que realizam comunicações que usam as memórias *Flash* presentes em ambas as placas serão introduzidos os procedimentos para o seu acesso.

Faça-se uma referência final ao facto de que a utilização de todas as iterações de envio ou leitura de dados e endereço do modelo apresentado não serem implementados em todos os projetos. Isto acontece porque as memórias que são endereçadas não têm uma dimensão que o justifique. No caso da memória interna os projetos endereçam um número reduzido de posições de memória, exclusivamente para validação do mecanismo de transferência de dados, totalizando 256 posições de 32 bits cada. No caso das memórias externas da placa Nexys 2 e da *Flash* presente na Atlys, todas com 128 Mbits, é possível realizar o endereçamento de forma alinhada ao Byte. No primeiro caso são suficientes 8 bits de

endereço e no segundo são suficientes 24 bits, algo que se reflete nos projetos implementados.

Quando aos comandos de escrita e leitura dos diversos projetos que usam comunicação USB-EPP, e que dependem de aplicações de consola operando num PC, deverão ser utilizados da seguinte forma:

- Operação de escrita de uma palavra:
  "'executavel.exe' –p XXX –d YYY –b WWW"
  onde "XXX" representa o endereço onde a palavra será escrita, "YYY" representa o dispositivo alvo (podendo ser "DOnbUsb" ou "Atlys" para Nexys 2 e Atlys respetivamente, por omissão) e "WWW" representa o valor de até 32 bits que se pretende escrever;
- Operação de leitura:
  "'executavel.exe' –g XXX –d YYY"
  onde os valores são idênticos aos apresentados para a operação de escrita;
- Operação de leitura de várias palavras de um ficheiro:
  "'executavel.exe' –s XXX –d YYY –b WWW –f ZZZ"
  onde "XXX" representa o endereço inicial onde se começarão a escrever as palavras, "WWW" é a quantidade de palavras a escrever e "ZZZ" representa o nome do ficheiro do qual serão lidas as palavras a escrever na memória alvo;
- Operação de escrita num ficheiro de várias palavras presentes na memória alvo:
  "'executavel.exe' –r XXX –d YYY –b WWW –f ZZZ" onde os valores são idênticos aos já apresentados para a operação de leitura de várias palavras de um ficheiro.

Conclui-se assim a apresentação do procedimento genérico de comunicação entre PC e FPGA usando o serviço de atendimento a interrupções disponível no microprocessador MicroBlaze, e que implementa um protocolo USB-EPP.

4.4.2 – Implementação de microprocessador Microblaze na placa Digilent Nexys 2

Nas duas secções seguintes apresentar-se-ão os projetos que implementam o microprocessador MicroBlaze e os núcleos necessários ao controlo das diversas memórias disponíveis na Nexys 2. Antes de apresentar os projetos que permitem a comunicação com um PC através de USB introduzem-se os projetos de controlo das memórias externas através da UART.

4.4.2.1 – Projeto de controlo das memórias externas *Flash* e SDRAM

Este projeto visa a implementação de diversas operações disponíveis para a utilização da *Flash*. Para tal desenvolveram-se funções simples que implementam os protocolos e

procedimentos descritos no *datasheet* do dispositivo [101]. As funções projetadas podem ser invocadas usando a ligação UART para acesso a um menu de controlo, não estando neste projeto disponível a ligação por USB. No entanto, e porque este tipo de memória externa é do tipo estático, ou seja, retêm os dados mesmo após ser retirada energia à placa, estas funções deverão revelar-se úteis não só para transferência de dados entre *Flash* e microprocessador como para armazenamento de longa duração. Caso se pretenda a recuperação dos dados para o PC deverá ser feita a sua leitura através da aplicação Adept 2, usando a secção do GUI adequada para o efeito, ou usando o protocolo UART implementado.

De entre as funções úteis à operação desta memória não volátil implementadas incluem-se:

- Função de escrita de dados;
- Função de leitura de dados;
- Função de leitura do Registo de Estado;
- Função de eliminação de dados de um setor da memória;
- Função de eliminação de todos os dados da *Flash*.

Tal como mencionado previamente, estas operações de acesso dependem de uma sequência de operações, algumas das quais usando as funções da biblioteca "xio" já mencionada. No entanto, para as realizar é necessária a escrita em endereços adequados de palavras de comando adequadas à operação que se pretenda, sendo em alguns casos necessária a escrita de palavras de comando adicionais ou a leitura do Registo de Estado da memória *Flash*. As funções implementadas podem ser lidas no anexo I.

No caso desta memória, os testes iniciais apresentavam valores de retorno díspares dos esperados, disponibilizados no *datasheet*. Realizou-se então uma série de operações de mapeamento da resposta da *Flash* a todas as palavras de comando possíveis: dado que o canal de dados desta memória é de 16 bits enviaram-se todas palavras possíveis até 16 bits e estudou-se a resposta obtida. Desse estudo resultou a conclusão de que a ligação entre o microprocessador MicroBlaze e o periférico se encontra revertida ao Byte, ou seja, o bit 1 para o microprocessador está mapeado no bit 8 do periférico, o bit 2 no bit 7 e de forma similar para os restantes bits, repetindo-se a situação no Byte seguinte.

No sentido de obviar esta dificuldade implementou-se uma lista estática de conversão de palavras de tamanho Byte. Após esta introdução as operações passaram a ser realizadas com sucesso.

Como apoio às funções implementadas projetaram-se funções de leitura através da UART de endereços, de quantidade de dados a ler ou escrever e de palavras a escrever na memória *Flash*.

Introduza-se finalmente uma característica das memórias *Flash* em estudo em ambas as placas. De facto a escrita destes espaços de armazenamento de dados é realizada através da mudança de estado de cada célula de bit de '1' para '0'. Assim, o procedimento de eliminação dos dados da memória é realizado colocando todas as células com valor '1' e o procedimento de escrita comuta apenas as células que correspondem aos bits '0' da palavra a escrever, como apresentado na figura 99.

Figura 99 - Operação de escrita em células Flash

Caso se realize uma operação de escrita numa área de memória onde previamente tenham sido escritos dados não é garantida a validade dos dados escritos, tal como patente na figura 100.

Figura 100 - Operação de escrita numa célula previamente preenchida com dados

Assim, e em resumo, a escrita em áreas da *Flash* previamente usadas para armazenamento deverá sempre ser precedida de operações de eliminação de dados, para que os dados escritos sejam válidos. No caso da *Flash* presente na Nexys 2, essa eliminação de dados só é possível em setores de 128 KBytes.

O código criado para este projeto está presente no anexo J.

O projeto que usa a memória externa SDRAM tem função similar ao projeto que usa a *Flash*, permitindo operações de escrita e leitura da memória SDRAM, teste e eliminação de dados.

Este projeto encontra-se igualmente limitado no que toca à recuperação dos dados, sendo possível realizar a sua escrita ou leitura através da UART. Realizar estas operações através da área do GUI da aplicação Adept 2 adequada ao efeito revelou-se não funcional: para além dos erros introduzidos na leitura de uma memória vazia, apresentando dados inexistentes, as mesmas condições de troca de bits verificadas para a Flash verificam-se neste ponto. Dado que é impossível a recuperação de dados através de Adept 2, não se introduziram mecanismos de reposição de bits na posição correta, pois as operações de escrita/leitura de dados usando a UART ou USB-EPP através de uma aplicação de consola não apresentam erros.

As operações são realizadas sobre a memória SDRAM usando as funções disponibilizadas pela biblioteca "Xio". O código que implementa as diversas funções, por ser virtualmente igual ao estudado para a memória DDR2-SDRAM presente na Atlys não será referido neste ponto, mas na seção apropriada dos projetos MicroBlaze implementados na Atlys.

4.4.2.2 – Transferência de dados entre PC e memórias presentes na placa Nexys 2

O primeiro projeto dos três que se descrevem de seguida, implementados na Nexys 2, permite o envio e receção de dados entre PC e blocos RAM da FPGA. O mecanismo de transmissão de palavras de 32 bits corresponde àquele apresentado na secção 4.4.1. Dada a disponibilidade de uma memória de acesso rápido na placa Nexys 2, a memória SDRAM, optou-se pela limitação a endereços de 8 bits criando uma área endereçável de 1 KByte de memória interna, com o intuito de testar a validade e funcionamento desta implementação. É possível adicionar memória interna alocando blocos RAM embutidos, recorrendo à aplicação Xilinx EDK para alterar o *hardware* instanciado até ao limite disponível na FPGA. Os projetos criados terão que partilhar esse espaço de endereçamento pelo código a implementar (na forma do ficheiro ELF), pela *heap* e pela *stack* do sistema.

Dado que neste projeto o endereço de armazenamento pode ser enviado para a FPGA ou recebido num único ciclo de leitura/escrita, o funcionamento do atendimento a interrupções USB-EPP apresenta alterações em relação ao modelo. A mais relevante será a redução do número de ciclos de leitura: no primeiro ciclo de leitura é entregue todo o endereço, pelo que os dados podem ser recuperados e parcialmente enviados na mesma iteração. O total de iterações para leitura e escrita são desta forma limitados pela dimensão da palavra de dados de 32 bits a quatro iterações para os dois tipos de operação possíveis. O código criado para o efeito pode ser parcialmente lido no anexo K, sendo apresentada a secção crítica do código, o algoritmo que é executado aquando de um serviço de atendimento à interrupção

O segundo projeto compreende os mesmos procedimentos estendidos à memória SDRAM. Dado o espaço de armazenamento endereçável com 24 bits, o mecanismo de serviço a interrupções está configurado para receção de três Bytes de endereço. As funções de leitura e escrita são aquelas já apresentadas, presentes na biblioteca "Xio". O código realizado para efeito de escrita e leitura pode ser visto no anexo L, uma vez mais de forma reduzida ao serviço de atendimento a interrupções.

O projeto que se apresenta de seguida, o terceiro desta secção, pretende a leitura e escrita na memória *Flash*. De dimensão igual à da memória SDRAM, as iterações para escrita e leitura de dados são as mesmas. No entanto as funções de acesso diferem e portanto implicam um protocolo de acesso diferente: enquanto é possível usar funções de escrita e leitura de 32 bits no caso SDRAM, no caso *Flash* as funções de escrita e leitura da biblioteca criada para o efeito apenas permitem a utilização de operadores de 16 bits.

Escrita em SDRAM		
Endereço	**Dados**	**Estado**
Byte 1 de 3	Byte 1 de 4	
Byte 2 de 3	Byte 2 de 4	
Byte 3 de 3	Byte 3 de 4	Endereço recebido
	Byte 4 de 4	Dados recebidos: escrita da palavra no endereço

Tabela 8 - Etapas de uma operação de escrita de dados na SDRAM

A tabela 8 apresenta os procedimentos já mencionados para a escrita em SDRAM. Verifica-se a receção de um endereço de três Bytes e de uma palavra de quatro Bytes, em que ambos podem ser utilizados como parâmetros de invocação da função de escrita da biblioteca "xio.h": "XIo_Out32()".

Escrita em *Flash*		
Endereço	**Dados**	**Estado**
Byte 1 de 3	Byte 1 de 4	
Byte 2 de 3	Byte 2 de 4	Primeiros 16 bits da palavra de dados recebidos
Byte 3 de 3	Byte 3 de 4	Endereço recebido: escrita dos primeiros 16 bits
	Byte 4 de 4	Segundos 16 bits da palavra de dados recebidos: escrita dos segundos 16 bits

Tabela 9 - Etapas de uma operação de escrita de dados na *Flash*

A tabela 9 apresenta os procedimentos para o caso *Flash*. As funções da biblioteca criada "flash.ops.c" permitem escrita de dois Bytes, sendo a palavra recebida separada e escrita em duas iterações: a primeira quando o endereço é recebido na totalidade; a segunda quando a totalidade da palavra é recebida.

Leitura da SDRAM		
Endereço	**Dados**	**Estado**
Byte 1 de 3		
Byte 2 de 3		
Byte 3 de 3	Byte 1 de 4	Endereço recebido: leitura dos dados
	Byte 2 de 4	
	Byte 3 de 4	
	Byte 4 de 4	Dados totalmente enviados

Tabela 10 - Etapas de uma operação de leitura de dados da SDRAM

No caso de operações de leitura os procedimentos são apresentados nas tabelas 10 e 11. As operações obrigam novamente ao fracionamento da palavra de dados nos procedimentos de leitura da *Flash*, sendo possível a leitura de meia palavra de 32 bits em cada iteração de leitura. O código de transferência de dados usando USB-EPP, entre PC e a

memória *Flash*, pode ser lido no anexo M e a biblioteca usada já havia sido apresentada no anexo I.

Leitura da *Flash*		
Endereço	**Dados**	**Estado**
Byte 1 de 3		
Byte 2 de 3		
Byte 3 de 3	Byte 1 de 4	Endereço recebido: leitura dos primeiros 16 bits
	Byte 2 de 4	Primeiros 16 bits enviados
	Byte 3 de 4	Leitura dos segundos 16 bits
	Byte 4 de 4	Segundos 16 bits enviados: dados totalmente enviados

Tabela 11 - Etapas de uma operação de leitura de dados da *Flash*

4.4.3 – Implementação de microprocessador Microblaze na placa Digilent Atlys

Nas duas próximas secções abordaremos a implementação de diversos projetos baseados em MicroBlaze usando a placa Digilent Atlys, projetos esses com paralelo nos projetos já apresentados para a placa Nexys 2. Assim será dada ênfase aos aspetos diferenciadores.

De forma similar à realizada para a placa Nexys 2, apresentaremos primeiro os projetos sem ligação USB-EPP e de seguida aqueles que a possuem.

4.4.3.1 – Projeto de controlo das memórias externas QuadSPI-Flash e DDR2-SDRAM

Estes dois projetos visam a implementação das diversas operações disponíveis para a utilização da QuadSPI-Flash e da DDR2-SDRAM nas bibliotecas Digilent e Xilinx.

De facto é disponibilizado um *Built In Self-Test* na página da Digilent referente à Atlys que inclui exemplos de invocação de funções que atuam sobre estas memórias externas [102]. Uma das bibliotecas, aquela que se foca no uso da QuadSPI-Flash, é disponibilizada na documentação que acompanha o IP *Core* desta memória periférica e este projeto visa a implementação exaustiva das funções fornecidas. No caso da memória DDR2-SDRAM os procedimentos são em larga medida similares aos usados para a memória SDRAM presente na Nexys 2.

De entre as funções que endereçam a QuadSPI-Flash que se distinguem das estudadas previamente refiram-se uma função de teste desta memória não volátil e uma função que permite a mudança de modo de operação para um de três disponíveis: *QuadSPI*, *DualSPI* e *ExtendedSPI*.

Em ambos os casos, e de forma idêntica aos projetos implementados para a Nexys 2, a única forma de comunicação externa depende da utilização do bloco UART pelo que este se revela indispensável à execução de qualquer operação.

Os códigos criados para estes projetos estão presentes nos anexos N e O para o caso QuadSPI-Flash e DDR2-SDRAM respetivamente.

4.4.3.2 – Transferência de dados entre PC e memórias presentes na placa Atlys

Os diversos projetos seguem a mesma linha dos projetos que usam USB-EPP e que foram implementados na Nexys 2. Atente-se portanto nas alterações que se revelaram necessárias para o funcionamento adequado aos periféricos presentes na Atlys.

Ignorando o projeto que endereça a memória interna, já apresentado para o caso Nexys 2, refira-se projeto para Atlys que endereça a memória DDR2-SDRAM. Este permite o acesso a 1 Gbit de armazenamento sendo portanto necessários 27 bits de endereço e portanto quatro ciclos para receber a totalidade do endereço. O código criado é, apesar dessa diferença, muito similar ao do anexo L pelo que será evitada a sua repetição.

A operação com *Flash* obrigou à criação de um projeto comparativamente diferente, considerando os projetos previamente apresentados tanto para Nexys 2 como para Atlys.

De facto a QuadSPI-Flash tem um acesso para escrita ou leitura de características diferentes daquelas que os projetos anteriores apresentam: são utilizadas funções específicas de acesso, desenhadas para escrita ou leitura de páginas por oposição a palavras, funções que realizam diversas operações durante o procedimento de escrita ou leitura de dados. Estas funções estão presentes na biblioteca fornecida pela Digilent.

A primeira iteração deste projeto foi implementada apoiando-se nas funções “Page Program” e “Fast Read”. Assim, e num protocolo similar àquele aplicado para a escrita de dados de 32 Bit na memória RAM externa, testou-se o envio da palavra e a invocação da função “Page Program” dos quatro Bytes recebidos. Refira-se que o volume de Bytes a escrever na QuadSPI-Flash que é possibilitado através desta função é um dos seus parâmetros de entrada.

Os resultados obtidos não foram os esperados: a recuperação dos dados da memória *Flash* falhava ciclicamente à quarta palavra de 32 bits em cada quatro palavras escritas, estando portanto corrupta. Considere-se que a operação “Page Program” permite a escrita de até 256 Bytes, ou seja 64 palavras de 32 Bit. A operação de leitura permite a leitura do mesmo volume de dados, 256 Bytes, e não apresentava falhas em ler apenas uma palavra de 4 Bytes.

Dada a dificuldade em isolar o motivo para a falha cíclica presente nas funções da biblioteca disponibilizada pela Digilent, sendo possível tratar-se de uma colisão devido aos *timings* das operações, e motivado pelos recursos dessas funções para a escrita e leitura de mais do que uma palavra, introduziram-se as seguintes alterações na área do código destinado à escrita de dados através de USB-EPP, exemplificado na figura 101.

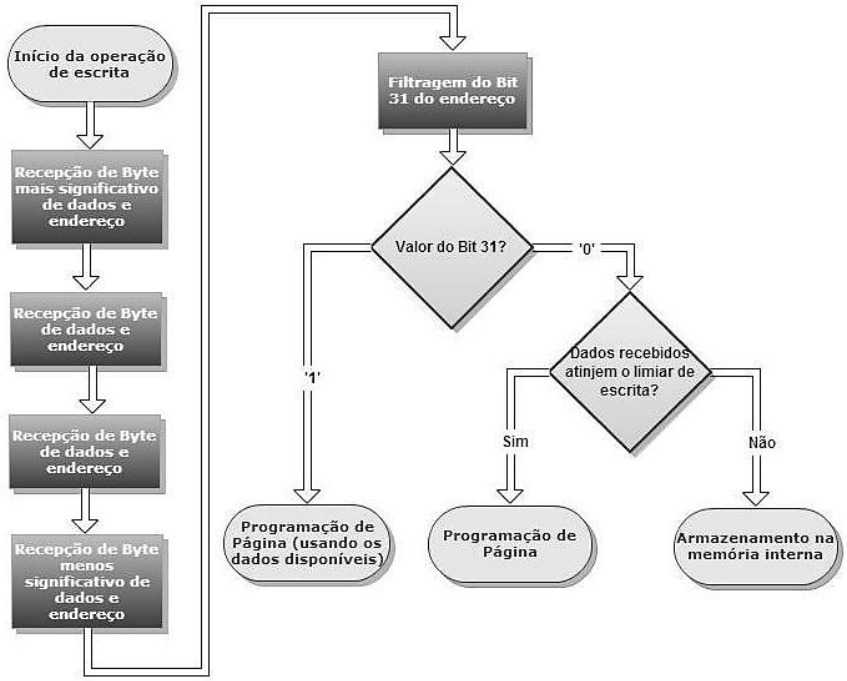

Figura 101 - Operação de escrita na QuadSPI-Flash

As operações realizadas são as seguintes:

- Aloca-se memória interna, durante a iniciação do projeto programado na FPGA, para armazenamento de até 32 palavras de 4 Bytes, ou seja 128 Bytes;
- Executa-se o procedimento genérico de receção de dados e endereços;
- Filtra-se o endereço recebido durante o processo de escrita para verificação do bit mais significativo:
    - Caso o Bit tenha valor '1', o processo de escrita de dados conclui com a presente palavra e portanto a função "Page Program" será invocada para escrita dos dados presentes na memória interna e da palavra recebida nessa iteração;
    - Caso o Bit tenha valor '0' e a área alocada na memória interna não fique preenchida com a palavra acabada de receber, alocar a palavra na memória interna e aguardar mais palavras;
    - Caso o Bit tenha valor '0' e a área alocada na memória interna fique preenchida com a palavra acabada de receber para escrita, invocar a função "Page Program" dos dados presentes na memória interna incluindo a palavra acabada de receber.

Este mecanismo foi desenhado para a escrita de 128 Bytes (meia palavra) pelos motivos apontados previamente relativos a falhas de *timing*, igualmente verificados para a escrita do volume máximo de 64 palavras de quatro Bytes. O valor escolhido representa um compromisso entre o limite máximo testado que a função permite (48 palavras de quatro Byte) e valores reduzidos que obrigassem a um número elevado de invocações da função.

Nestas condições resta assegurar a sinalização do último elemento a enviar para programação. Tal foi conseguido alterando a aplicação de consola que opera no PC. Se no caso

da escrita de uma palavra individual basta proceder à adição desse valor ao bit 31 do endereço (sem risco de corrupção de dados pois o espaço de endereçamento só requer 27 bits), no caso da leitura de um ficheiro externo para a memória QuadSPI-Flash este exercício prova ser mais complexo.

De facto a leitura usando a aplicação de consola de um ficheiro externo para escrita na QuadSPI-Flash é limitada por dois grandes elementos: o total de valores de quatro Bytes a ler passado como parâmetro e o volume de dados efetivamente presente no ficheiro externo. Como a última palavra é enviada associada a um endereço sinalizado, caso se atingisse o final do ficheiro a última palavra enviada já teria sido armazenada na memória interna da FPGA e, quebrado o ciclo de leitura, todas as palavras armazenadas nesse espaço não seriam escritas na memória externa.

Para obviar este problema procede-se à pré-leitura de uma palavra do ficheiro externo com dados, realizando a leitura das linhas seguintes presentes no ficheiro dentro de um ciclo. Esse ciclo de envio realiza à cabeça leitura de nova linha do ficheiro externo, armazena-o temporariamente e procede ao envio do valor imediatamente anterior. Desta forma é possível identificar se o ciclo irá ser quebrado por ter chegado ao fim do ficheiro e, em caso afirmativo, sinalizar esse fim colocando o Bit 31 do endereço a '1' na palavra prestes a enviar.

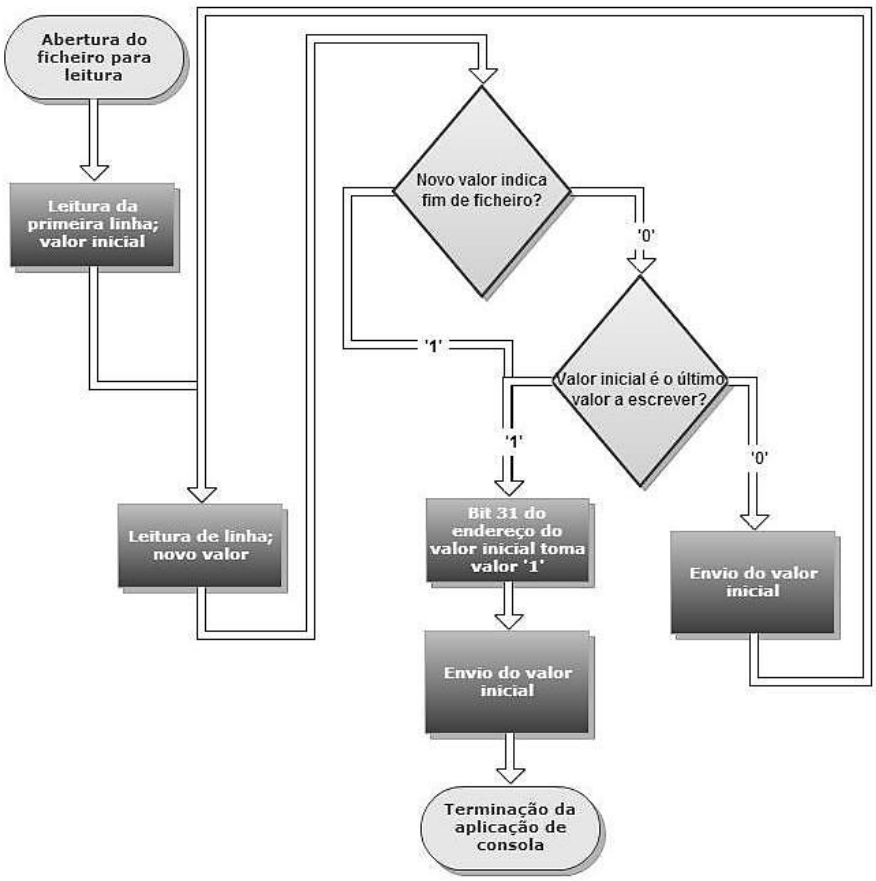

Figura 102 - Operação de leitura de dados de ficheiro externo para a QuadSPI-Flash

O mecanismo é ilustrado pelo diagrama de fluxo de dados da figura 102. Analise-se o processo de envio:

- É realizada a abertura do ficheiro para leitura de dados e realizada a leitura da primeira linha antes de entrar no ciclo de leitura;
- Dentro do ciclo é realizada nova leitura de linha: caso seja sinalizado fim de ficheiro, soma o endereço associado ao valor inicial o bit 31 a '1' (máscara 0x80000000); em caso negativo armazena-se o novo valor e avança-se para o teste seguinte;
- Caso o valor inicial represente a palavra final a enviar soma ao endereço associado a essa palavra o Bit 31 de valor '1' e procede ao seu envio; em caso negativo não altera o endereço e envia os dados;
- É realizada nova leitura de linha: o valor armazenado passa a valor inicial e o valor acabado de ler na nova linha é avaliado, repetindo o ciclo.

O código criado para este projeto está presente nos anexos P e Q, onde o anexo P apresenta um extrato do código implementado em MicroBlaze e o anexo Q um extrato do código usado para criar a aplicação de consola que opera no PC.

O projeto final é motivado pela problemática introduzida nos parágrafos anteriores, associadas à escrita de dados na QuadSPI-Flash, e também pela necessidade de apagar a área da *Flash* onde se pretenda efetivamente escrever dados para que esta operação seja realizada com fiabilidade.

A memória externa em questão pode ser apagada de duas formas diferentes: eliminando um subsetor ou eliminando todo um setor, com capacidades de 4 KBytes e 64 KBytes respetivamente. A análise do *datasheet* da memória *Flash* apresenta um tempo típico para conclusão das operações de 0,2 e 0,7 segundos, claramente acima do tempo de *timeout* da operações do core USB-EPP, onde o sinal de WAIT não pode ficar bloqueado em nenhum nível durante as operações de leitura e escrita por tempo superior a 0,01 segundos [103][104].

Considerou-se então o seguinte:

- Que os projetos apresentados se destinam a permitir comunicações com a placa Digilent Atlys e o preenchimento das memórias disponíveis com dados;
- Que representam o ponto de arranque para o desenvolvimento de novos projetos e implementações desenvolvidas para o microprocessador MicroBlaze;
- Que a disponibilidade no mesmo projeto de ambas as memórias externas e da UART baseada em USB torna possível o controlo completo das operações realizadas sobre ambas, beneficiando das características de elevados débitos de transmissão da memória DDR2-SDRAM e a não volatilidade dos dados escritos na memória QuadSPI-Flash.

Dadas estas condições, e considerando a dimensão muito superior da memória externa DDR2-SDRAM comparativamente à memória QuadSPI-Flash, optou-se pela realização do *backup* dos dados da segunda numa área reservada da primeira. A reserva da memória é assegurada pela limitação da área endereçável de todas as operações realizáveis, tanto na aplicação programada através do ficheiro "ELF" na FPGA como na aplicação de consola em

operação no Computador, usada para enviar e receber dados através de USB-EPP. A figura 103 ilustra a disposição implementada.

Começando pelo endereço mais baixo endereçável da DDR2-SDRAM (0x0):

- A primeira área de memória é memória não reservada, disponível para operações de escrita e leitura similares às já estudadas para memórias RAM externas;
- Seguidamente, começando no endereço 0x06FEFF00, pode encontrar-se a área que mapeia o *backup* da QuadSPI-Flash em setores; caso seja realizada alguma operação sobre os dados de um dado setor do *backup*, quer tenham origem no serviço USB-EPP quer tenham origem nas funções internas controladas por UART, este mapa atribui valor positivo '1' para o sinalizar; é assim indicado que os dados presentes no *backup* e os dados presentes na QuadSPI-Flash diferem naquele setor; em caso contrário o mapa atribui valor '0';
- A área seguinte atribui valor positivo '1' ou nulo '0' como forma de indicar o estado de ocupação das páginas da memória *Flash*, caso tenha dados ou não tenha dados respetivamente;
- A área final é o *backup* dos dados que se encontram na QuadSPI-Flash, transferidos graças a uma operação automática realizada nos procedimentos iniciais após a programação da FPGA.

Figura 103 - Mapa de ocupação da memória externa DDR2-SDRAM

Através deste conjunto de áreas de memória, que inclui mapa de ocupação e atualização, é possível a obtenção de diversos benefícios em relação aos projetos apresentados previamente e que incluem:

- Escrita indireta na QuadSPI-Flash, passando primeiro pela área de *backup* na DDR2-SDRAM o que origina transferências de dados comparativamente mais rápidas;
- A eliminação de setores necessária à escrita de novos dados de forma segura é realizada de forma automática, garantindo a fiabilidade dos dados escritos na QuadSPI-Flash;
- É possível a visualização rápida da ocupação da memória *Flash*.

No sentido de suportar estas operações, a aplicação que corre no microprocessador MicroBlaze e a aplicação de consola operando no PC, e que realiza a transferência de dados, possuem funções de filtragem dos endereços que impedem operações de escrita ou leitura indevida.

As duas operações mais relevantes são identificadas como "Store Data In QuadSPI *Flash* (post USB operations)" e "Retrieve Data From QuadSPI *Flash* (pre USB operations)" no menu de operações. A primeira deverá realizar-se caso se pretenda atualizar os dados presentes na QuadSPI-Flash: esta operação percorre o mapa de setores da área de backup e, cada vez que identifica um setor sinalizado com valor positivo '1', apaga esse setor na memória *Flash* e programa as páginas desse setor com os dados presentes no backup, como ilustrado na figura 104.

A segunda poderá realizar-se a seguir à execução da operação de teste da memória DDR2-SDRAM, dado que esta operação é destrutiva, alterando os dados presentes em diversas áreas dessa memória inclusive na área reservada ao backup. Correndo esta operação executa-se uma série de operações "Fast Read" que copia os dados presentes na *Flash*.

Os procedimentos encontram-se igualmente ilustrados na figura 104.

Para além das funções de escrita para atualização dos dados na QuadSPI-Flash e leitura para atualização do backup presente na memória DDR2-SDRAM, estão presentes funções que permitem:

- Apagar setores físicos;
- Apagar toda a memória *Flash*;
- Ler, apagar e escrever dados pseudo-aleatórios ou dados do utilizador em ambas as memórias externas.
- Operações de eliminação de setores ou da totalidade da memória *Flash* que atualizam automaticamente o estado do *backup*.

Fica assim concluída a apresentação deste projeto que estará no topo da hierarquia dos vários apresentados, pela possibilidade de leitura e escrita em ambas as memórias externas presentes na placa Atlys.

O código criado para este projeto está presente nos anexos R e S, onde o anexo R apresenta o código integral implementado no MicroBlaze e o anexo S o código integral implementado usando Visual Studio 2012 numa aplicação de consola.

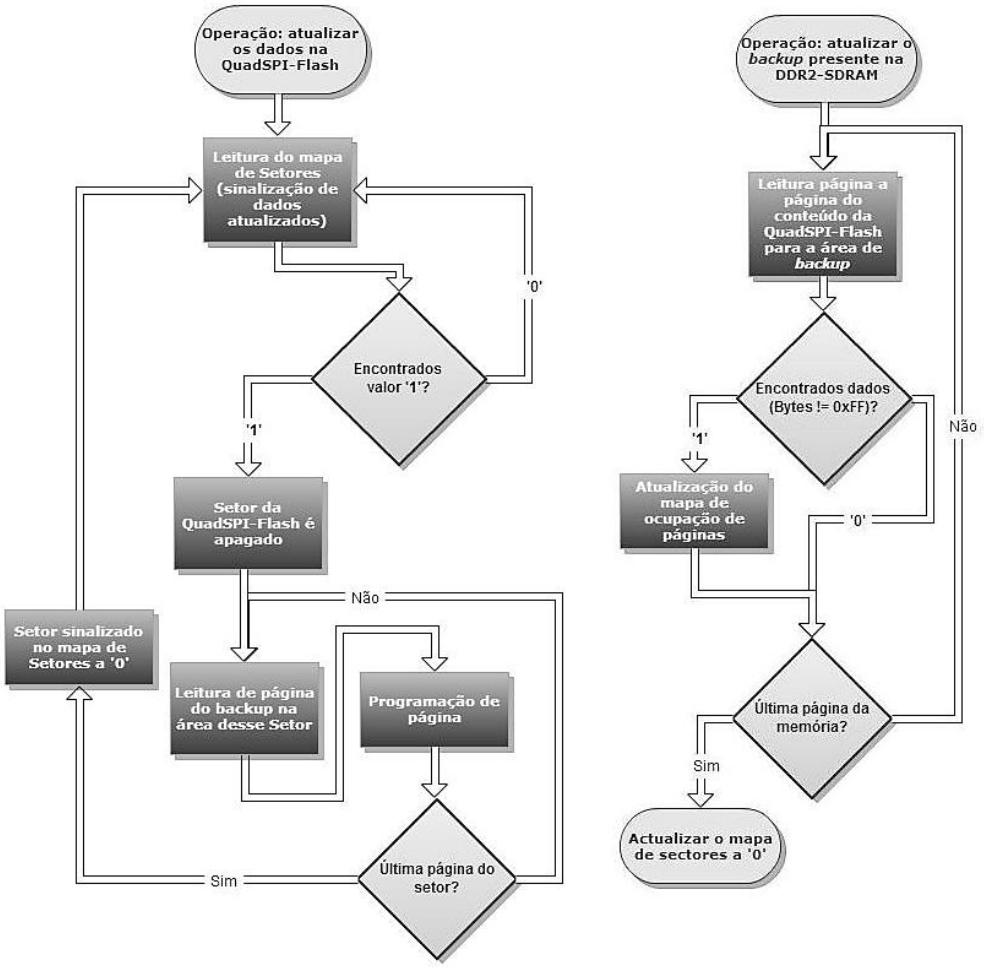

Figura 104 - Operações de refrescamento de QuadSPI-Flash e do backup na DDR2-SDRAM

4.5 – Conclusão

Este capítulo introduziu as diversas ferramentas criadas e as características das suas implementações. Os projetos abordados foram:

- Uma aplicação geradora de dados pseudo-aleatórios;
- Uma aplicação conversora de ficheiros binários
- Uma aplicação comparadora de dados;
- Uma aplicação geradora de ficheiros COE;
- Um projeto de envio e receção de dados de 32 bits entre Computador Pessoal e blocos RAM da FPGA, usando o protocolo USB-EPP;
- Vários projetos de envio e receção de dados de 32 bits entre Computador Pessoal e as diversas memórias presentes na placa Atlys, usando o protocolo USB-EPP apoiado no microprocessador MicroBlaze.

No próximo capítulo far-se-á uma breve discussão dos resultados e uma apresentação do trabalho futuro que poderá ser feito sobre os projetos aqui introduzidos.

Capítulo 5 – Resultados

Este capítulo documenta os resultados associados aos projetos de transferência de dados desenvolvidos, cobrindo duas vertentes: os recursos ocupados por cada projeto e as taxas de transferência obtidas.

Começando por abordar a ocupação de recursos do projeto mapeado na lógica da FPGA, a tabelas 12 apresenta resultados para a Nexys 2 e a Atlys, obtidos a partir dos relatórios pós síntese dos ficheiros de programação (Xilinx ISE – Project Navigator). Os valores apresentados representam todos os elementos da arquitetura da FPGA cuja ocupação é não nula com exceção dos blocos RAM, omitidos dado que são um recurso configurável pelo utilizador em função das necessidades da implementação pretendida.

Recursos com ocupação não nula mais relevantes		
	Nexys 2	**Atlys**
Registos	1%	1%
LUT	1%	1%
Slices ocupadas	1%	1%
BUFIO2/BUFIO2_2CLKs	-	3%
BUFIO2FB/BUFIO2FB_2CLKs	-	3%
BUFG/BUFGMUX	8%	6%
DCM/DCM_CLKGENs	12%	12%

Tabela 12 - Ocupação de recursos - Interface USB-EPP

Os elementos da malha lógica ocupados totalizam em ambas as placas um número reduzido e os valores reportados de 1% para ambas as placas não refletem a mesma escala de ocupação, dada que a FPGA presente na Atlys apresenta uma malha lógica com mais blocos lógicos. São ainda apresentados valores baixos para os elementos de gestão e síntese dos sinais de relógio. Estes resultados são os esperados dada a simplicidade dos protocolos de comunicação que a interface USB-EPP implementa.

Passando aos recursos ocupados pelos diversos projetos realizados com apoio no microprocessador MicroBlaze, as diversas configurações realizadas tiveram como objetivo a obtenção de sistemas que permitissem a transmissão de dados de teste de forma fiável. Estas configurações, apresentadas já na secção 3.2.1.2.1, não exploram todas as ferramentas disponibilizadas para a redução dos recursos ocupados exatamente por esse motivo. No entanto, e dada a premissa de que estes sistemas de transferência funcionem como apoio a outros projetos a implementar na lógica da FPGA, a quantidade de elementos da malha lógica ocupados foi reduzido através da remoção de algumas opções de elevado desempenho deste microprocessador. Adicionalmente, os blocos RAM alocados para a implementação das áreas de código, *heap* e *stack* entre outros, foram dimensionados de forma a permitirem o carregamento dos ficheiros "ELF" (onde o código C a implementar se encontra compilado) na configuração que permite a realização de *debug* (por norma de tamanho superior àqueles obtidos na configuração *release*). As tabelas 13 e 14 apresentam os valores obtidos para a placa Nexys 2 e Atlys respetivamente.

Ocupação de recursos das implementações MicroBlaze na placa Digilent Nexys 2					
Elementos:	***Flash***	**SDRAM**	**USB**	**USB e SDRAM**	**USB e *Flash***
Flip-flops	9%	14%	12%	18%	14%
Slices ocupadas	25%	49%	31%	61%	35%
LUT	16%	41%	22%	48%	24%
RAMB16	57%	57%	67%	67%	67%
BUFGMUX	8%	8%	12%	12%	12%
DCM	12%	12%	12%	12%	12%
BSCAN	100%	100%	100%	100%	100%
MULT18X18SIO	10%	10%	10%	10%	10%

Tabela 13 - Ocupação de recursos - MicroBlaze - Nexys 2

Ocupação de recursos das implementações MicroBlaze na placa Digilent Atlys						
Elementos:	**QuadSPI *Flash***	**DDR2 SDRAM**	**USB**	**USB e DDR2 SDRAM**	**USB e QuadSPI *Flash***	**USB, DDR2 SDRAM e QuadSPI *Flash***
Flip-Flops	3%	4%	4%	5%	5%	6%
LUT	8%	10%	10%	14%	12%	16%
Slices ocupadas	14%	17%	16%	22%	21%	28%
MUXCY	2%	2%	2%	3%	3%	4%
RAMB16BWER	27%	31%	29%	32%	29%	32%
RAMB8BWER	1%	0%	1%	1%	1%	1%
BUFG/BUFGMUX	12%	12%	18%	18%	18%	18%
ILOGIC2/ISERDES2	0%	1%	2%	2%	2%	2%
IODELAY2/IODRP2/IODRP2_MCB	0%	6%	0%	6%	0%	6%
OLOGIC2/OSERDES2	1%	11%	0%	11%	1%	13%
BUFPLL_MCB	0%	25%	0%	25%	0%	25%
DSP48A1	5%	5%	5%	5%	5%	5%
MCB	0%	50%	0%	50%	0%	50%

Tabela 14 - Ocupação de recursos - MicroBlaze - Atlys

Nestas implementações, as gerações diferentes a que as FPGA Xilinx pertencem fica patente nos resultados obtidos. A ocupação da malha lógica da Spartan-3E é sempre elevada, com valores para as slices ocupadas a começaram nos 25% e a subirem até aos 61%, assim como é a ocupação de blocos RAM. Os resultados para o caso da Spartan 6 apresentam uma diferença notoriamente favorável. Repare-se na tabela referente à Atlys que os elementos de

transferência de dados de memórias de alto débito apresentam ocupação nos projetos que envolvem a DDR2-SDRAM.

Os resultados apresentados de seguida focam-se nas taxas de transferência obtidas para os diversos projetos.

Os projetos mapeados na lógica da FPGA apresentados na secção 4.3 funcionaram como referência para os diversos débitos testados devido à implementação mais simples, onde a lógica implementada é exclusivamente dedicada ao interface, e à presença de um sintetizador de relógio que fornece um sinal facilmente configurável à máquina de estados finitos síncrona. Foram testadas diversas frequências de relógio em ambas as placas, até ao limite recomendado pela ferramenta de mapeamento e encaminhamento da aplicação Xilinx ISE – *Project Navigator*, e diversos volumes de dados:

- Na Nexys 2 foram testadas frequências entre 25 e 165 MHz e volumes de dados de 2048, 8192 e 16384 valores de 32 bits;
- Na Atlys 2 foram testadas frequências entre 25 e 220 MHz e os mesmos volumes de dados.

Foi ainda introduzido em todos os projetos um mecanismo simples que impedisse a inicialização e funcionamento da UART pois, apesar de vantajosa para efeitos de *debug* durante o desenvolvimento, o seu funcionamento revelou-se penalizador para os débitos de transferência de dados testados.

Os resultados obtidos pautaram-se por variações muito reduzidas apesar de as frequências testadas abrangerem um espectro considerável. De facto, tomando como exemplo os resultados obtidos para a Nexys 2, os desvios padrão inferiores a 1% e 3% para o caso da operação de escrita e leitura respetivamente, sublinham como os resultados obtidos pertencem a um intervalo muito reduzido. Os valores, que podem ser encontrados na tabela 15 na linha "Sem MicroBlaze", são muito baixos considerando os valores típicos das transferências USB 2.0 (usualmente da ordem dos MBytes/segundo).

Valores médios das taxas de transferência (valores em KBytes/segundo)					
Placa de prototipagem		**Nexys 2 -1200**		**Atlys**	
Operação realizada		**Escrita**	**Leitura**	**Escrita**	**Leitura**
Sem MicroBlaze	Bloco RAM	1,59	1,28	1,6	1,27
Com MicroBlaze	Bloco RAM	1,13	1,15	1,15	1,14
	RAM	1,37	0,93	1,31	0,72
	Flash	1,4	0,92	1,02	0,59

Tabela 15 - Valores médios das taxas de transferência dos projetos desenvolvidos

No caso dos diversos projetos realizados usando MicroBlaze foram testados os mesmos volumes de dados mas não foi realizada qualquer alteração ao sinal de relógio adotado pelo sistema. Os resultados são igualmente baixos em relação aos valores típicos USB 2.0 e comparativamente inferiores, podendo ser analisados nas linhas "Com MicroBlaze" da tabela 15.

Considerem-se para análise desta área da tabela os seguintes dados:

- O número de iterações necessárias para a realização da leitura de dados nos casos das memórias externas é superior ao necessário no projeto sem MicroBlaze;
- O número de operações de I/O necessárias à conclusão da escrita/leitura de dados no caso da QuadSPI-Flash presente na Atlys é comparativamente elevado;
- O projeto de escrita/leitura nos blocos RAM, usando MicroBlaze, foi desenvolvido para efeitos de teste e validação do protocolo USB-EPP usando o sistema de atendimento a interrupções; não tendo sido alocados blocos RAM em maior quantidade (pois tal penalizaria a ocupação deste recurso), o volume de valores testados foi reduzido.

Conclui-se desta forma a apresentação dos resultados mais relevantes obtidos para os projetos de transferência de dados desenvolvidos.

Capítulo 6 – Conclusões e Trabalho Futuro

Os projetos desenvolvidos têm como objetivo o funcionamento como suporte para projetos futuros e nesse sentido são introduzidas algumas ferramentas funcionais com aplicação prática imediata nas placas Digilent testadas.

A primeira ferramenta criada é a geradora de dados pseudo-aleatórios, com distribuição uniforme e com 32 bits de informação. Esta cria com sucesso diversos ficheiros úteis a operações de ordenação de dados, incluindo listas de dados no formato decimal e hexadecimal ASCII e também binário.

Procede ainda à ordenação dos dados gerados no formato decimal e hexadecimal usando um algoritmo eficiente e rápido, disponibilizando assim um termo de comparação para dados que sejam ordenados numa FPGA, por exemplo. Os ficheiros binários criados usando esta aplicação têm como alvo o carregamento dos dados gerados nas memórias *Flash* das placas Digilent testadas.

A segunda ferramenta criada é conversora desses ficheiros binários para formatos de fácil leitura. Destinada a ser utilizada sobre os dados lidos das memórias *Flash* após o seu processamento, esta ferramenta lê com sucesso os ficheiros e procede à criação de uma lista com os dados lidos, nos formatos decimal e hexadecimal ASCII.

A terceira ferramenta criada endereça as necessidades de operações de ordenação de dados e é uma ferramenta comparadora que lê duas listas de valores, usualmente a lista de valores ordenados num Computador Pessoal e a lista de valores ordenada numa FPGA, e valida a igualdade ou identifica as diferenças entre ambas, potenciando a validação da implementação em FPGA que esteja a ser testada.

Estes projetos apresentam grande simplicidade de utilização, sendo necessários poucos argumentos para a execução de cada um. A sua utilização permite a disponibilização rápida de ficheiros com dados pseudo-aleatórios de 32 bits, a conversão dessas listas de e para o formato binário e a comparação entre os dados gerados e outros processados.

O projeto seguinte é uma aplicação geradora de ficheiros COE. O seu funcionamento é aquele projetado pois apresenta grande flexibilidade cobrindo as diversas aplicações deste tipo de ficheiros. De facto a execução deste projeto produz listas de dados para utilização por diversos IP *Core* Xilinx, tendo sido testado com sucesso através do carregamento na memória interna das FPGA. A possibilidade de cobrir os vários tipos de blocos que precisam de ficheiros de coeficientes, as várias bases em que os dados podem ser apresentados e a dimensão variável das palavras geradas atestam das capacidades desta aplicação de consola.

No entanto, esta grande flexibilidade e capacidade obrigam à entrada através da linha de comandos de um número de parâmetros elevado, em particular se se pretender a introdução manual de dados para escrita do ficheiro. O procedimento apresenta-se demasiado simples, sendo nesta fase impossível realizar correções se forem introduzidos erros na escrita manual. Este projeto beneficiaria bastante da introdução de um ambiente gráfico que permitisse uma configuração mais simples e visual e também a possibilidade de controlar os dados visualmente durante as operações através da utilização de rato, por exemplo.

O terceiro grande projeto estudado escreve e lê da memória interna das FPGA dados de 32 bits, tanto introduzidos manualmente através da linha de comandos como por leitura de um ficheiro externo. Representando uma implementação que constrói sobre um mecanismo de comunicação destinado à escrita e leitura manual de valores de dimensão Byte, a extensão a 32 bits de dados e até 16 bits de endereço mediante a utilização de vários ciclos de envio e receção prova ser funcional, possibilitando o carregamento dos blocos RAM das FPGA presentes na Nexys 2 e na Atlys durante a operação de *hardware* já implementado na sua lógica. Este nível de liberdade é muito superior àquele oferecido pela geração de ficheiros COE e sua leitura e implementação, sendo a operação de envio ou receção de dados mais rápida que a execução de todos os passos do processo de programação de uma FPGA.

Se os resultados de ocupação de recursos apresentam valores reduzidos, permitindo um grau de liberdade elevado ao futuro utilizador para a implementação de projetos de grande dimensão na lógica da FPGA de ambas as placas, as taxas de transferência obtidas são bastante reduzidas. A motivação para este resultado, em face dos testes realizados, deverá encontrar-se nas bibliotecas e funcionalidades associadas à aplicação Adept 2 que implementam este protocolo no PC e que não foram alteradas. Apesar de representar um método mais rápido de transferência de dados para os blocos RAM do que o carregamento de um ficheiro "COE" pré-síntese do ficheiro de programação da FPGA, não deixa de se encontrar longe das capacidades que a ligação USB 2.0 permite.

Projetos futuros nos quais seja crucial um carregamento mais rápido de dados poderão debruçar-se sobre o funcionamento dos *drivers* e da aplicação de consola que envie ou receba os dados. Um exemplo proposto para substituir o pacote Adept 2 pode ser encontrado *online* de forma gratuita e, apesar de não oferecer um GUI apelativo, refere taxas de transferência bastante superiores: a ferramenta FPGALink, compatível com ambas as placas testadas [105].

Quanto aos projetos realizados para MicroBlaze, as funcionalidades disponibilizadas replicam em larga medida as funcionalidades implementadas pela aplicação Adept 2, de controlo das placas Digilent. É possível o carregamento de dados e sua leitura nas memórias externas não voláteis e a eliminação dos dados presentes nessas memórias. Os projetos apresentam no entanto vantagens adicionais que os tornam proveitosos em relação à utilização da aplicação citada.

De facto as operações mencionadas implicam que a aplicação Adept 2 programe a FPGA com *hardware* de controlo desses periféricos de armazenamento, efetivamente interrompendo as operações e eliminando a implementação que se encontrasse a operar. Por outras palavras, o acesso às memórias não acontece de forma simultânea com qualquer outra implementação na FPGA que não aquela dedicada à transferência desses dados.

Os projetos realizados obviam este problema dado que é possível a escrita ou leitura de dados em pontos aleatórios das memórias, de volumes de dados de dimensão aleatória. Adicionalmente, é possível a escrita e leitura de dados na DDR2-SDRAM presente na Atlys, a maior área de armazenamento nas duas placas testadas e um periférico de acesso rápido. Usando os mecanismos de comunicação disponibilizados pelo MicroBlaze, destinados a permitir a comunicação com *hardware* implementado pelo utilizador na FPGA, por exemplo um núcleo de ordenação, será possível a utilização das diversas memórias externas como suporte às operações realizadas por esse *hardware*.

As taxas de transferência obtidas para a escrita e leitura de dados nas diversas memórias, quer interna quer externa à FPGA, são igualmente baixas. Enquanto alguns resultados são expectáveis, como a taxa de transferência comparativamente inferior obtida para o projeto que endereça a QuadSPI-Flash presente na Atlys, os resultados obtidos no projeto que escreve ou lê na DDR2-SDRAM encontram-se abaixo dos esperados. Se por um lado a operação de leitura da última necessita de um maior número de iterações, dado o espaço de endereçamento maior de entre todas as memórias externas estudadas, a operação de escrita deveria apresentar uma taxa superior, dado que é realizada num mesmo número de iterações que as restantes memórias e endereça aquela que é a memória com maior potencial para elevados débitos de dados. A avaliação das alterações a realizar aos diversos projetos encontra-se dependente da alteração da ferramenta de transferência de dados no anfitrião, tal como já concluído para o projeto mapeado na lógica da FPGA sem o apoio do microprocessador MicroBlaze.

Apesar do sucesso na construção dos diversos projetos é relevante que se atente sobre os recursos ocupados para as diversas implementações. Se no caso da Atlys os recursos são relativamente reduzidos, no caso da Nexys 2 isso não se verifica. Sendo o objetivo destes projetos permitir a implementação de projetos futuros, na forma de núcleos de hardware implementáveis na lógica da FPGA, poderá revelar-se difícil ou mesmo impossível criar uma implementação cuja dimensão não ultrapasse as capacidades da Spartan-3E em estudo.

Considerando que o objetivo desta Dissertação não foi a implementação de soluções orientadas à menor área possível, mas apenas garantir fiabilidade e estabilidade dos processos de I/O de dados, o trabalho a jusante poderá passar por projetos adaptados a microprocessadores *Soft Core* que ocupem uma área lógica inferior, como o PicoBlaze [106]. Este é ideal para situações em que se pretende a utilização mínima de recursos, podendo ser considerada uma versão do MicroBlaze onde grande parte das suas funcionalidades de elevado desempenho não estão presentes.

Como alternativa poderá considerar-se não implementar um microprocessador *Soft Core* na FPGA, optando por programar no *hardware* os controladores adequados à comunicação com a memória interna e com as memórias externas disponíveis, numa opção que deverá apresentar o menor consumo de recursos nesta FPGA limitada. A Digilent oferece no caso da Nexys 2 implementações que permitem a comunicação com ambas as memórias externas presentes nessa placa [107]. No entanto a implementação fornecida depende da aplicação de consola para um correto funcionamento, requerendo que o utilizador programe os atrasos entre comunicações e as palavras de comando necessárias para efetivamente enviar ou receber dados, não representando portanto na sua forma atual uma solução eficiente ou fiável.

No caso da Atlys não são disponibilizados projetos que exemplifiquem a utilização das memórias externas sem depender de um microprocessador MicroBlaze. O trabalho a jusante poderá passar pelo desenvolvimento de um IP *Core* que implemente o protocolo QuadSPI que a *Flash* presente na Atlys usa, assim como a implementação, teste e validação do IP *Core* de gestão de memórias disponibilizado pela Xilinx na aplicação ISE – Project Planner (para utilizar a memória DDR2-SDRAM). Existem alguns avanços na configuração precisa deste IP *Core*, indispensável para beneficiar das elevadas taxas de transferência que esta memória externa permite, todos disponíveis online [108][109][110].

Em síntese projetaram-se nesta Dissertação ferramentas de geração e de transferência de dados entre um computador de uso pessoal e uma placa Digilent Nexys 2 ou Digilent Atlys.

Referências

[1] http://www.theinquirer.net/inquirer/news/1811460/fpga-manufacturer-claims-beat-moores-law

[2] http://www.altera.com/b/innovating-at28-nm.html

[3] http://www.altera.com/corporate/news_room/releases/2012/products/nr-optical-fpga-demo.html

[4] http://www.design-reuse.com/articles/18692/fpga-video-processing-platform.html

[5] http://www.cotsjournalonline.com/articles/view/100910

[6] http://www.eetimes.com/design/military-aerospace-design/4216729/Xilinx-FPGAs-beam-up-next-gen-radio-astronomy

[7] http://www.rtcmagazine.com/articles/view/100227

[8] Ian Grout, *Digital Systems Design With FPGA And CPLDs*, Newnes-Elsevier, ISBN 978-0-7506-8397-5, pp.17-21, 2008.

[9] Richard S. Sandige and Michael L. Sandige, *Fundamentals of Digital and Computer Design with VHDL*, McGraw-Hill, ISBN 978-0-07-338069-8, pp. 212, 2012.

[10] Richard S. Sandige and Michael L. Sandige, *Fundamentals of Digital and Computer Design with VHDL*, McGraw-Hill, ISBN 978-0-07-338069-8, pp. 213, 2012.

[11] Richard S. Sandige and Michael L. Sandige, *Fundamentals of Digital and Computer Design with VHDL*, McGraw-Hill, ISBN 978-0-07-338069-8, pp. 214, 2012.

[12] Richard S. Sandige and Michael L. Sandige, *Fundamentals of Digital and Computer Design with VHDL*, McGraw-Hill, ISBN 978-0-07-338069-8, pp. 210-214, 2012.

[13] http://www.ustudy.in/node/7586

[14] Peter J. Ashenden, *Digital Design – An Embedded Systems Approach Using VHDL*, Morgan Kaufman, ISBN 978-0-12-369528-4, pp. 274-275, 2008

[15] http://eda360insider.wordpress.com/2011/10/27/3d-thursday-generation-jumping2-5d-xilinx-virtex7-2000t-fpga-delivers1954560-logic-cells-using6-8-billion-transistors/

[16] http://www.xilinx.com/about/company-overview/index.htm

[17] Jean-Pierre Deschamps, Gery Jean Antoine Bioul and Gustavo D. Sutter, *Synthesis Of Arithmetic Circuits - FPGA, ASIC, and Embedded Systems*, John Wiley & Sons, Inc., ISBN 978-0471-68783-2, pp. 258, 2006.

[18] http://www.ami.ac.uk/courses/ami4460_fpga/u02/#2.1.2

[19] Clive "Max" Maxfield, *FPGAs World Class Designs*, pp. 17-18, www.newnespress.com.

[20] http://www.eetimes.com/design/eda-design/4213740/STOP--Are-You-Gambling-On-Your-Memory-IP

[21] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN 0-7506-7604-3, pp. 61-65, 2004.

[22] http://chipsmagic.blogspot.pt/2010/03/sram-static-random-access-memory-sram.html

[23] Clive "Max" Maxfield, *FPGAs World Class Designs*, pp. 14-15, www.newnespress.com.

[24] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=244435

[25] http://www.elec.york.ac.uk/staff/ajg112.html

[26] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN 0-7506-7604-3, pp. 66-68, 2004.

[27] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=244435

[28] http://www.pcbdesign007.com/pages/columns.cgi?clmid=17&artid=47570&_pf_=1

[29] http://www.ustudy.in/node/7618

[30] Clive "Max" Maxfield, *FPGAs World Class Designs*, pp. 20-22, www.newnespress.com

[31] http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[32] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=242565

[33] Clive "Max" Maxfield, *FPGAs World Class Designs*, pp. 23-27, www.newnespress.com

[34] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=244435

[35] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN 0-7506-7604-3, pp. 78-79, 2004.

[36] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=244435

[37] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN 0-7506-7604-3, pp. 79-80, 2004.

[38] http://www.design-reuse.com/articles/19106/systolic-fir-filter-based-fpga.html

[39] Clive "Max" Maxfield, *FPGAs World Class Designs*, pp. 141, www.newnespress.com

[40] http://www.eetimes.com/design/programmable-logic/4014849/FPGA-Architectures-from-A-to-Z--Part2

[41] http://www.eetimes.com/design/programmable-logic/4014849/FPGA-Architectures-from-A-to-Z--Part2

[42] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN 0-7506-7604-3, pp. 80-83, 2004.

[43] Ricardo Reis, Marcelo Lubaszewski, Jochen A.G. Jess, *Design of Systems on a Chip*, Springer, ISBN 978-0-387-32499-9, pp. 220, 2006.

[44] http://www.programmableplanet.com/author.asp?section_id=2154&doc_id=246185

[45] http://www.xilinx.com/tools/microblaze.htm

[46] http://www.impulseaccelerated.com/xilinx/impulse_kit_mb.html

[47] Milica Mitic and Mile Stojcev, *An Overview of On-Chip Buses*, ELEC. ENERG. vol. 19, no. 3, Faculty of Electronic Engineering, University of Nis, Nis, Serbia, December 2006, pp. 405-428, http://facta.junis.ni.ac.rs/eae/fu2k63/stojcev.pdf

[48] http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf

[49] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=244435

[50] http://asic.co.in/Index_files/fpga_interview_questions.htm

[51] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN: 0-7506-7604-3, pp. 84-89, 2004.

[52] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=244435

[53] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN 0-7506-7604-3, pp. 89-92, 2004.

[54] http://www.programmableplanet.com/author.asp?section_id=1925&doc_id=244435

[55] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*, Newnes-Elsevier, ISBN 0-7506-7604-3, pp. 92-93, 2004.

[56] http://www.xilinx.com/products/intellectual-property/index.htm

[57] Jean-Pierre Deschamps, Gery Jean Antoine Bioul and Gustavo D. Sutter, *Synthesis Of Arithmetic Circuits - FPGA, ASIC, and Embedded Systems*, John Wiley & Sons, Inc., ISBN 978-0471-68783-2, pp. 255, 2006.

[58] Jean-Pierre Deschamps, Gery Jean Antoine Bioul and Gustavo D. Sutter, *Synthesis Of Arithmetic Circuits - FPGA, ASIC, and Embedded Systems*, John Wiley & Sons, Inc., ISBN-13 978-0471-68783-2, pp. 255, 2006.

[59] Jean-Pierre Deschamps, Gery Jean Antoine Bioul and Gustavo D. Sutter, *Synthesis Of Arithmetic Circuits - FPGA, ASIC, and Embedded Systems*, John Wiley & Sons, Inc., ISBN 978-0471-68783-2, pp. 266, 2006.

[60] Jean-Pierre Deschamps, Gery Jean Antoine Bioul and Gustavo D. Sutter, *Synthesis Of Arithmetic Circuits - FPGA, ASIC, and Embedded Systems*, John Wiley & Sons, Inc., ISBN 978-0471-68783-2, pp. 264-266, 2006.

[61] http://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_p_smp_starting_project_using_xps_alone.htm

[62] http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm

[63] http://www.altera.com/b/stratix-v-fpga.html

[64] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/silicon-devices/index.htm

[65] http://media.corporate-ir.net/media_files/IROL/21/212763/XLNXCS588_EPP_Pele_Diagram_FINAL.jpg

[66] http://digilentinc.com/NavTop/AboutUs.cfm

[67] http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2

[68] http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS

[69] http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2

[70] http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[71] http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2

[72] http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS

[73] http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[74] http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS

[75] http://www.exar.com/connectivity/uart-and-bridging-solutions/usb-uarts/xr21v1414

[76] http://www.xilinx.com/products/design-tools/ise-design-suite/

[77] http://www.digilentinc.com/Data/Products/ATLYS/Atlys_rm.pdf

[78] http://www.xilinx.com/univ/hwboards_starter_features.htm

[79] http://www.xilinx.com/tools/platform.htm

[80] http://digilentinc.com/Data/Products/ATLYS/Atlys_BSB_Support_v_3_6.zip

[81] http://digilentinc.com/Data/Products/NEXYS2/Nexys2_1200_Edk_11_Ram_Flash.zip

[82] http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2

[83] http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2

[84] http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,768&Prod=DIGILENT-PLUGIN

[85] http://notepad-plus-plus.org/

[86] http://sourceforge.net/projects/npp-plugins/files/Hex%20Editor/

[87] http://frhed.sourceforge.net/en/

[88] http://www.ieeta.pt/~iouliia/Papers/2011/FPL.pdf

[89] http://dl.acm.org/citation.cfm?doid=272991.272995

[90] http://www.mcs.anl.gov/~kazutomo/hugepage-old/twister.c

[91] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms- Third Edition*, The MIT Press, ISBN 978-0-262-03384-8, pp. 170-184, 2009.

[92] http://www.cs.auckland.ac.nz/~jmor159/PLDS210/niemann/s_qui.htm

[93] http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/cgn_r_coe_file_syntax.htm

[94] http://www.digilentinc.com/Cart/Download.cfm?DURL=/Data/Products/adept2/digilent.adept.sdk_v2.1.1.zip&ProductID=AdeptSDK2.1.1-W

[95] Digilent Asynchronous Parallel Interface (DEPP), pp. 3, Digilent, 2010, http://www.digilentinc.com/Cart/Download.cfm?DURL=/Data/Products/adept2/digilent.adept.sdk_v2.1.1.zip&ProductID=AdeptSDK2.1.1-W

[96] Digilent Asynchronous Parallel Interface (DEPP), pp. 3, Digilent, 2010, http://www.digilentinc.com/Cart/Download.cfm?DURL=/Data/Products/adept2/digilent.adept.sdk_v2.1.1.zip&ProductID=AdeptSDK2.1.1-W

[97] Digilent Asynchronous Parallel Interface (DEPP), pp. 4, Digilent, 2010, http://www.digilentinc.com/Cart/Download.cfm?DURL=/Data/Products/adept2/digilent.adept.sdk_v2.1.1.zip&ProductID=AdeptSDK2.1.1-W

[98] https://github.com/makestuff/vhdl/tree/master/dpimext

[99] http://digilentinc.com/Data/Products/NEXYS2/Nexys2_1200General.zip

[100] http://digilentinc.com/Data/Products/ATLYS/AtlysGeneralUCF.zip

[101] http://www.micron.com/parts/nor-flash/parallel-nor-flash/js28f128j3f75h

[102] http://digilentinc.com/Data/Products/ATLYS/Atlys_ManTest3_InitMemTest_Clean.zip

[103] http://www.micron.com/parts/nor-flash/serial-nor-flash/n25q128a13bsfh0f

[104] http://www.digilentinc.com/Data/Products/ADEPT/DpimRef%20programmers%20manual.pdf

[105] http://www.makestuff.eu/wordpress/software/fpgalink/

[106] http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf

[107] http://digilentinc.com/Data/Documents/Reference%20Designs/OnBoardMemCfg.zip

[108] http://tristesse.org/FPGA/XilinxMIGTutorial

[109] http://tristesse.org/20110209

[110] http://xlnx.lithium.com/t5/MIG-Memory-Interface-Generator/having-IOB-error-while-implementa-ting-DDR2-design-on-Atlys/td-p/244276

ANEXO A

Gerador de Dados Pseudo-aleatórios

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

#define _CRT_SECURE_NO_WARNINGS
#define MAX_ELEMENTS 8388607
#define ATLYS_MAX_ELEMENTS 4194303
#define MAX_ATLYS_SIZE 16777216
#define MASKA 0xFF000000
#define MASKB 0x00FF0000
#define MASKC 0x0000FF00
#define MASKD 0x000000FF
#define RANDMASK 0x7FFF

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>
#include <math.h>
#include "quicksort.c"             //for insertion or quicksorting
#include "twistergen.c"            //for 32 bit random values using
Mersenne Twister Pseudo-Random Number Generator

void quickSort(T *a, tblIndex lb, tblIndex ub);
void seedMT(uint32 seed);
uint32 randomMT(void);

void main (int argc, char *argv[])
{
      // Variables definition
      char readfilename[FILENAME_MAX-14];
      char filename[FILENAME_MAX];
      char filenamesorted[FILENAME_MAX];
      //char filenameforde2[FILENAME_MAX];
      char filenameforflash[FILENAME_MAX];
      char filenamedecimal[FILENAME_MAX];
      char filenamedecsort[FILENAME_MAX];
      FILE *dataFile;
```

```
      FILE *dataFileSorted;
      //FILE *dataFileAltera;
      FILE *dataFileXilinx;
      FILE *dataFileDecimal;
      FILE *dataFileDecSort;

      int bytesNum;
      int dataItemsNum;
      int i;
      unsigned int *dataList, *dataListSort;
      int MASKFILLER = 0xFFFFFFFF;
      char atlyslim;

      // Description
      printf("\n");
    printf("Usage: DataGEN FILENAME NUMBER_OF_DATA_ITEMS\n");
    printf("This program will create 4 files:\n");
    printf("\t1. With name = FILENAME.txt; There you will find
NUMBER_OF_DATA_ITEMS\n");
      printf("\t\tnon repeating randomly generated numbers. \n");
    printf("\t\tEach number appears on new line and is represented
in\n");
      printf("\t\thexadecimal 8 digit form.\n");
    printf("\t2. With name = FILENAME_sorted.txt; There you will find the
same data\n");
      printf("\t\tpresented in the same way, but sorted.\n");
    //printf("\t3. With name = FILENAME_for_DE2.hex; You can use this
file to store\n");
      //printf("\t\tgenerated data to DE2-115 flash. Use starting address
0.\n");
    printf("\t3. With name = FILENAME_for_Flash.bin; You can use this
file to store\n");
      printf("\t\tgenerated data to Atlys flash. Use starting address
0.\n");
       printf("\t4. With name = FILENAME_dec.txt; There you will find
NUMBER_OF_DATA_ITEMS\n");
      printf("\t\tnon repeating randomly generated numbers in decimal
format.\n");
       printf("\t5. With name = FILENAME_dec_sorted.txt; There you will
find the same data\n");
      printf("\t\tpresented in the same way, but sorted.\n");
    printf("WAIT UNTIL the \"Finished.\" message is printed. Until that
moment files are still\n");
      printf("\t\tbeing generated.\n");
    printf("\n");

      // argument # check
      if (argc != 3) {
            printf("Wrong parameters. Use 2 parameters.\n");
        exit(1);
    }
```

```
	//filenames setup
	strcpy(readfilename,argv[1]);
	strcpy(filename,readfilename);
	strcat(filename,".txt");
	strcpy(filenamesorted,readfilename);
	strcat(filenamesorted,"_sorted.txt");
	//strcpy(filenameforde2,readfilename);
	//strcat(filenameforde2,"_for_DE2.hex");
	strcpy(filenameforflash,readfilename);
	strcat(filenameforflash,"_for_Flash.bin");
	strcpy(filenamedecimal,readfilename);
	strcat(filenamedecimal,"_decimal.txt");
	strcpy(filenamedecsort,readfilename);
	strcat(filenamedecsort,"_decsorted.txt");

	//create and setup files to write to
	dataFile          = fopen(filename, "w");
	dataFileSorted    = fopen(filenamesorted, "w");
	//dataFileAltera  = fopen(filenameforde2, "w");
	dataFileXilinx    = fopen(filenameforflash, "wb");
	dataFileDecimal = fopen(filenamedecimal, "w");
	dataFileDecSort = fopen(filenamedecsort, "w");


	if((dataFile == NULL)|(dataFileSorted == NULL)|/*(dataFileAltera ==
NULL)*/(dataFileXilinx == NULL)|(dataFileDecimal ==
NULL)|(dataFileDecSort == NULL)){
		printf("Cannot open one of the output files\n");
		goto Fail;
	}

	// write number of bytes to be read (address limit)
	dataItemsNum= atoi(argv[2]);

	//check value validity
	if (dataItemsNum > MAX_ELEMENTS) {
		printf("Requested NUMBER_OF_DATA_ITEMS is very high,\n");
		printf("\tand above the safety limit for this operation. Now
exiting...\n");
		goto Fail;
	}

	//check value validity
	if (dataItemsNum > ATLYS_MAX_ELEMENTS) {
		printf("Requested NUMBER_OF_DATA_ITEMS exceeds 4194304,\n");
		printf("\twhich is the limit that fits in Nexys 2 and
Atlys\n");
		printf("\tflash memory, do you wish to proceed? (Y or N)\n");
		do {
			scanf ("%c\n", &atlyslim);
```

```
                if (atlyslim == 'y') atlyslim = 'Y';
                if (atlyslim == 'n') atlyslim = 'N';
                printf("Pressed key is not mapped for this op, try
again! (Y or N)\n");
            } while ((atlyslim != 'Y')|(atlyslim != 'N'));

            if (atlyslim == 'N') goto Fail;
      }


      bytesNum = dataItemsNum*4 + 4;
      putc((bytesNum&MASKA)>>24,dataFileXilinx);
      putc((bytesNum&MASKB)>>16,dataFileXilinx);
      putc((bytesNum&MASKC)>>8,dataFileXilinx);
      putc((bytesNum&MASKD),dataFileXilinx);
      //fprintf(dataFileAltera,"%08X",bytesNum);


      // large arrays memory allocation
      dataList = (unsigned int*)malloc(sizeof(unsigned
int)*dataItemsNum);
      if (dataList == NULL)   {
            printf("Failed allocating memory\n");
            exit(1);
      }

      // large arrays memory allocation
      dataListSort = (unsigned int*)malloc(sizeof(unsigned
int)*dataItemsNum);
      if (dataListSort == NULL)    {
            printf("Failed allocating memory\n");
            exit(1);
      }

      seedMT((unsigned int)time(NULL));

      // data gen
      for (i=0;i<dataItemsNum;i++){
            dataList[i]=(unsigned int)(0xFFFFFFFF&randomMT());
            dataListSort[i]=dataList[i];
      }


      printf("Writing %d 32bit data items = %d / 0x%08X bytes (4 bytes
per data item).\n",dataItemsNum,bytesNum,bytesNum);

    //write data items to files
    for (i = 0; i < dataItemsNum; i++) {
            fprintf(dataFile,"%08X\n",dataList[i]);
            fprintf(dataFileDecimal,"%u\n",dataList[i]);
            //fprintf(dataFileAltera,"%08X",dataList[i]);
```

```
            putc((dataList[i]&MASKA)>>24,dataFileXilinx);
            putc((dataList[i]&MASKB)>>16,dataFileXilinx);
            putc((dataList[i]&MASKC)>>8,dataFileXilinx);
            putc((dataList[i]&MASKD),dataFileXilinx);
      }

      // fill rest of the file for atlys flash with ff
    for (i=0;i<(MAX_ATLYS_SIZE-bytesNum);i++){
            putc(MASKFILLER,dataFileXilinx);
      }

      //Quicksort here
      quickSort(dataListSort, 0, dataItemsNum);

      //write sorted data to file
      for (i = 0; i < dataItemsNum; i++) {
            fprintf(dataFileSorted,"%08x\n",dataListSort[i]);
            fprintf(dataFileDecSort,"%u\n",dataListSort[i]);
      }

      // closing info
    /*printf("Use these numbers for reading back data from DE2-115
flash.\n");
    printf("\tADDRESS: %08X\n",bytesNum / 2);
    printf("\tLENGTH: %08X\n", (bytesNum + 5) / 2);
    printf("\nIn case of Atlys or Nexys 2 Flash you can read only entire
flash.\n"); */
      printf("\n");
    printf("Finished.\n");

      //cleanup routines


Fail: if ((dataFile!= NULL) & (dataFileSorted!= NULL) &
/*(dataFileAltera!= NULL)*/ (dataFileXilinx!= NULL) & (dataFileDecimal!=
NULL) & (dataFileDecSort!= NULL))  {
                  fclose(dataFile);
                  fclose(dataFileSorted);
                  //fclose(dataFileAltera);
                  fclose(dataFileXilinx);
                  fclose(dataFileDecimal);
                  fclose(dataFileDecSort);
            }

}
```

ANEXO B

Conversor de ficheiros binários para lista de dados em texto

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

#define _CRT_SECURE_NO_WARNINGS
#define MAX_ATLYS_SIZE 16777216
#define MAX_ELEMENTS 4194303

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void convertData(char *fileName, int numBytesToSkip, int
numBytesToRead);

void main(int argc,char * argv[]) {

      int bytesNumToRead;
      int bytesNumToSkip;
      char fileName[FILENAME_MAX-15];

      //Discription
     printf("Usage: DataCONV FILENAME NUMBER_OF_DATA_ITEMS\n");
     printf("This program will convert the file that was read
back\n");
      printf("from flash memory (named FILENAME.bin) into a
text\n");
      printf("file (FILENAME_converted.txt).\n");
     printf("Data in this text file will be represented as 4 byte
data items,\n");
      printf("each data item on a new line.\n");
      printf("Two versions will be produced so they can be compared
to the files\n");
      printf("with sorted data produced by DataGEN.\n");
     printf("\n");

      //Argument # check
     if (argc != 3) {
         printf("Wrong parameters.\n");
             exit(1);
```

```
          }

            //filename setup
          strcpy(fileName,argv[1]);

            //number of elements to convert setup
          bytesNumToRead = atoi(argv[2]);
            if (bytesNumToRead >= MAX_ELEMENTS){
                  printf("This file doesn't have those many
elements.\n");
                  printf("Application will now close.\n");
                  exit(1);
            }
            bytesNumToRead = bytesNumToRead * 4;
          bytesNumToSkip = bytesNumToRead + 4;

            //the process itself
            convertData(fileName, bytesNumToSkip, bytesNumToRead);

            //closing
          printf("\n");
          printf("Finished.\n");

      }

      void convertData(char *fileName, int numBytesToSkip, int
numBytesToRead){

            char fileNameOriginal[FILENAME_MAX];
            char fileNameConverted[FILENAME_MAX];
            char fileNameConvertedDec[FILENAME_MAX];
            FILE *dataFileOriginal;
            FILE *dataFileConverted;
            FILE *dataFileConvertedDec;
            unsigned int curvalue = 0,finalvalue = 0;
            int i;

            strcpy(fileNameOriginal,fileName);
            strcat(fileNameOriginal,".bin");
            strcpy(fileNameConverted,fileName);
            strcat(fileNameConverted,"_converted.txt");
            strcpy(fileNameConvertedDec,fileName);
            strcat(fileNameConvertedDec,"_converted_decimal.txt");

            //create and setup files to write to
            dataFileOriginal = fopen(fileNameOriginal, "rb");
            dataFileConverted = fopen(fileNameConverted, "w");
            dataFileConvertedDec = fopen(fileNameConvertedDec, "w");

                  if((dataFileOriginal == NULL)|(dataFileConverted ==
NULL)|(dataFileConvertedDec == NULL)){
```

```
                printf("Cannot open one of the data files\n");
                goto Fail;
          }
          for (i=0; i<4; i++){
                curvalue = getc(dataFileOriginal);  //Read and waste
first 4 Bytes with data amount

          }
          //data conversion
          for (i = 1; i < numBytesToRead+1; i++) {
                curvalue = getc(dataFileOriginal);
                finalvalue += curvalue;
                if (i % 4 != 0){
                      finalvalue <<= 8;
                }
                else {
                      fprintf(dataFileConverted,"%08X\n",finalvalue);
                      fprintf(dataFileConvertedDec,"%u\n",finalvalue);
                      finalvalue=0;
                }
          }

    // cleaning up and closing
    Fail: if ((dataFileOriginal!= NULL) & (dataFileConverted!= NULL) &
(dataFileConvertedDec != NULL))
                      {
                            fclose(dataFileOriginal);
                            fclose(dataFileConverted);
                            fclose(dataFileConvertedDec);
                      }
    }
```

ANEXO C

Comparador de listas de dados em texto

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

#define _CRT_SECURE_NO_WARNINGS
#define CHARS_PER_ELEMENT 11
#define notdiff 0

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void compare (char* fileName1,char* fileName2);

void main(int argc, char *argv[]) {

      char filename1[FILENAME_MAX-4];
      char filename2[FILENAME_MAX-4];

    printf("Usage: DataCOMP FILENAME1 FILENAME2 \n");
    printf("This program will compare 2 files with names FILENAME1.txt
and FILENAME2.txt\n");
    printf("If there are any differences this application will point to
the line\n");
      printf("in which differences appear.\n");
    printf("\n");

      if (argc != 3) {
            printf("Wrong parameters.\n");
        exit(1);
    }

      strcpy(filename1,argv[1]);
      strcpy(filename2,argv[2]);

      compare(filename1,filename2);

      printf("\n");
    printf("Finished.\n");
```

```
}

void compare (char* filename1,char* filename2){

      char teststring1[CHARS_PER_ELEMENT];
      char teststring2[CHARS_PER_ELEMENT];
      FILE *datafile1;
      FILE *datafile2;
      char readfilename1[FILENAME_MAX-4];
      char readfilename2[FILENAME_MAX-4];
      int lineNum = 0;
      int diffFound = 0;

      //core
      strcpy(readfilename1,filename1);
      strcpy(readfilename2,filename2);
      strcat(readfilename1,".txt");
      strcat(readfilename2,".txt");

    datafile1 = fopen(readfilename1, "r");
      datafile2 = fopen(readfilename2, "r");

      if((datafile1 == NULL)|(datafile2 == NULL)){
            printf("Cannot open one of the input files\n");
            goto Fail;
      }

      while ((!feof(datafile1)) & (!feof(datafile2))){
            lineNum += 1;
            fscanf(datafile1,"%s\n",&teststring1);
            fscanf(datafile2,"%s\n",&teststring2);
            if (strcmp(teststring1,teststring2)!=notdiff){
                  printf("Files differ on line %d: %s:%s !=
%s:%s\n",lineNum,filename1,teststring1,filename2,teststring2);
                  diffFound += 1;
            }
      }

      if (diffFound == 0){
            printf("Files %s and %s match.\n",filename1,filename2);
            printf("There were %d data items.\n",lineNum);
    }

Fail: if ((datafile1!= NULL) & (datafile2!= NULL)){
                  fclose(datafile1);
                  fclose(datafile2);
            }
}
```

ANEXO D

Gerador de ficheiros COE

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

#define _CRT_SECURE_NO_WARNINGS
#define S3E_500BRAM 368640
#define S6_45BRAM 2138112
#define BIN_STR_MAXSIZE 33
#define BITS_PER_HEXA_CHAR 4
#define FILTER(w)  (FILTERList[w])

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <ctype.h>
#include "twistergen.c"            //for  32  bit  random  values  using
Mersenne Twister Pseudo-Random Number Generator

void header(int tp, int rd, int rw, int rad, FILE *filein);
void makerandlist(int rd, int rw, unsigned int* list);
void workinput(int rd, int rw, int rad, unsigned int* list);
void printdata(FILE* fileout, int rd, int rw, int radix, unsigned int*
list);
void dec2binconv(unsigned int value,int ramwidth,char *binstr);
void seedMT(uint32 seed);
uint32 randomMT(void);

static unsigned int FILTERList[33] = {
      0x0,
      0x00000001, 0x00000003, 0x00000007, 0x0000000F,
      0x0000001F, 0x0000003F, 0x0000007F, 0x000000FF,
      0x000001FF, 0x000003FF, 0x000007FF, 0x00000FFF,
      0x00001FFF, 0x00003FFF, 0x00007FFF, 0x0000FFFF,
      0x0001FFFF, 0x0003FFFF, 0x0007FFFF, 0x000FFFFF,
      0x001FFFFF, 0x003FFFFF, 0x007FFFFF, 0x00FFFFFF,
      0x01FFFFFF, 0x03FFFFFF, 0x07FFFFFF, 0x0FFFFFFF,
      0x1FFFFFFF, 0x3FFFFFFF, 0x7FFFFFFF, 0xFFFFFFFF
};
```

```
void main(int argc,char * argv[]){

      FILE* datain;
      FILE* dataout;
      char filename[FILENAME_MAX], filenameout[FILENAME_MAX], temp;
      int ramdepth, ramwidth, radix, i, oper=0, type=0;
      unsigned int *valuelist;

      //Intro and option selection
      printf("\n\n\nThis exe is designed to produce a coe file, used in
BRAM initialization\n");
      printf("Target devices are Xilinx FPGA solutions with
BlockRAMs\n\n");
      printf("This app can either use values to fill BRAM from a
file(A),\n");
      printf("\tcreate a list of random values(B) or introduce one by
one(C)\n\n");
      printf("If a filename with txt extension is passed as argument\n");
      printf("\tthat file's contents will be used to create the coe
file;\n");
      printf("\tvalues should be placed one per line;\n");
      printf("If no filename is passed as argument\n");
      printf("\tvalues will be read from stdin\n\n");
      printf("Finally, the type of coe file desired (Memory, Coeficients,
...)\n");
      printf("\tcan also be selected.\n");

      printf("Info required is:\n");
      printf("\t1st - Desired operation\n");
      printf("\t2nd - Number of values to be written to the coe file (RAM
Depth)\n");
      printf("\t3rd - Number of characters per value (RAM Width)\n");
      printf("\t4th - Radix value: 2 (binary), 10 (decimal) or
16(hexadecimal)\n");
      printf("\t5th - Coe file desired\n");

      printf("\nChoose the operation intended:\n");
      printf("\t1 - Use the input file passed as argument for a values
source\n");
      printf("\t2 - Create a list of random values\n");
      printf("\t3 - Manually introduce values\n\t:");

      scanf("%d",&oper);
      if ((oper!=1)&(oper!=2)&(oper!=3)) {
            printf("Operation character selected is not valid\n");
            printf("Exiting app\n");
            exit(1);
      }

      printf("\tEnter RAM Depth:\n\t:");
```

```
	scanf("%d",&ramdepth);
	if ((ramdepth<1)|(ramdepth>(pow(2.0,24.0)))){
		printf("Number of values requested isn't within an acceptable
range\n");
		printf("Exiting app\n");
		exit(1);
	}


	printf("\tEnter RAM Width\n\t:");
	scanf("%d",&ramwidth);
	if((ramwidth<1)|(ramwidth>32)){
		printf("Value size isn't within an acceptable range\n");
		printf("Please choose between 1 and 32 bits\n");
		printf("Exiting app\n");
		exit(1);
	}

	if ((ramwidth*ramdepth) > S3E_500BRAM)	{	// check for size vs
Spartan-3E500
		printf("The requested combination of depth and value width
will not fit\n");
		printf("\tin Nexys 2 - Spartan-3E blockRAM. Do you wish to
continue?\n");
		do {
			printf(": ");
			scanf("%c", &temp);
			temp = tolower(temp);
			if ((temp != 'n') & (temp!= 'y')){
				printf("Value entered is not a mapped option, try
again.\n");
			}
		} while ((temp != 'n') & (temp!= 'y'));
		if (temp == 'n') exit (1);
	}

	if ((ramwidth*ramdepth) > S6_45BRAM)	{	// check for size vs
Spartan-6 SLX45 blkRAM
		printf("The requested combination of depth and value width
will not fit\n");
		printf("\tin Atlys - Spartan-6 blockRAM. Do you wish to
continue?\n");
		do {
			printf(": ");
			scanf("%c", &temp);
			temp = tolower(temp);
			if ((temp != 'n')&(temp!= 'y')){
				printf("Value entered is not a mapped option, try
again.\n");
			}
		} while ((temp != 'n')&(temp!= 'y'));
```

```
            if (temp == 'n') exit (1);
      }

      printf("\tEnter Radix Value\n\t:");
      scanf("%d",&radix);
      if((radix!=2)&(radix!=10)&(radix!=16)){
            printf("Radix value requested isn't an accepted value\n");
            printf("Please choose either 2, 10 or 16\n");
            printf("Exiting app\n");
            exit(1);
      }

      printf("Time to choose the kind of coe file; you have 5 types
available\n");
      printf("1 - COEFDATA\n");
      printf("\tCoefficients for a filter.\n");
      printf("2 - MEMORY_INITIALIZATION_VECTOR\n");
      printf("\tBlock and distributed memories.\n");
      printf("3 - PATTERN\n");
      printf("\tBit Correlator\n");
      printf("4 - BRANCH_LENGTH_VECTOR\n");
      printf("\tInterleaver\n");
      printf("5 - MEMDATA\n");
      printf("\tObsolete keyword (older implementations)\n");
      printf("\nChoose the type intended\n\t:");
      scanf("%d",&type);
      if ((type!=1)&(type!=2)&(type!=3)&(type!=4)&(type!=5)) {
            printf("Operation character selected is not valid\n");
            printf("Exiting app\n");
            exit(1);
      }

      //allocating memory for values
      valuelist = (unsigned int*)malloc(sizeof(unsigned int)*ramdepth);

      if (valuelist == NULL){
            printf("Failed allocating memory\n");
            exit(1);
      }

      //creating input and output streams
      if (oper==1){
            if (argc == 2){
                  strcpy(filename,argv[1]);
                  strcpy(filenameout,filename);
                  if((datain=fopen(filename,"r")) == NULL) {
                        printf("Error opening input file!\n");
                        printf("File with values should be named
datain.txt .\n");
                        exit (1);
                  }
```

```
                  strcpy(filenameout,strtok(filenameout,"."));
                  strcat(filenameout,".coe");
                  if ((dataout=fopen(filenameout,"w")) == NULL) {
                        printf("Error opening input file!\n");
                        printf("File   with   values   should   be   named
datain.txt .\n");
                        exit (1);
                  };
                  i=0;
                  while ((!feof(datain))&(i<ramdepth))     {
      //read file to get values

                        fscanf(datain,"%u\n", &valuelist[i]);
                        if     (valuelist[i]>     FILTER(ramwidth))     {
/*(pow(2.0,(long)ramwidth)))*/
                        //crop bit info above selected ramwidth
                              valuelist[i]                                =
FILTER(ramwidth)&valuelist[i]; //(int)(pow(2.0,(long)ramwidth))-1;

                        }
                        i++;
                  }
                  if (i<ramdepth) {
                        printf("External file only had %d values (below
the %d requested).\n", i, ramdepth);
                        ramdepth=i;
                  }
            }
            else{
                  printf("In order to open a file, the filename must be
passed as argument\n");
                  printf("Exiting app\n");
                  exit(1);
            }
      }
      else{
            if ((dataout=fopen("yourcoefile.coe","w")) == NULL) {
                  printf("Error opening input file!\n");
                  printf("File  with  values  should  be  named  datain.txt
.\n");
                  exit (1);
            };
      }

      //header insertion
      header(type,ramdepth,ramwidth,radix,dataout);

      //random values list generation
      if (oper==2){
            makerandlist(ramdepth,ramwidth,valuelist);
```

```
	}

	//alternatively read values from stdin
	else if (oper==3){
		workinput(ramdepth,ramwidth,radix,valuelist);
	}

	printdata(dataout,ramdepth,ramwidth,radix,valuelist);

	free(valuelist);
	printf("Completes free malloced array successfully\n"); //TESTING
	fclose(dataout);
	printf("Completes data out closing successfully\n"); //TESTING

	if ((oper==1)&(datain!=NULL)){
		fclose(datain);
	}

	printf("Task successfully completed\n");

	return;
}

void header(int tp,int rd,int rw, int rad,FILE *filein){

	switch(tp){
	case 1:
		fprintf(filein,";COE FILE\n");
		fprintf(filein,";\n");
		fprintf(filein,";The data that follows comprises the
coefficients of a filter.\n");
		fprintf(filein,";Specification for this file is:\n");
		fprintf(filein,";%d values with a %d bit width and base
%d.\n",rd,rw,rad);
		fprintf(filein,"radix = %d;\n",rad);
		fprintf(filein,"coefdata =\n");
		break;
	case 2:
		fprintf(filein,";COE FILE\n");
		fprintf(filein,";\n");
		fprintf(filein,";The data that follows comprises the contents
of a memory.\n");
		fprintf(filein,";Specification for this file is:\n");
		fprintf(filein,";%d values with a %d bit width and base
%d.\n",rd,rw,rad);
		fprintf(filein,"memory_initialization_radix = %d;\n",rad);
		fprintf(filein,"memory_initialization_vector =\n");
		break;
	case 3:
		fprintf(filein,";COE FILE\n");
		fprintf(filein,";\n");
```

```
            fprintf(filein,";The data that follows comprises the
coefficients of a bit correlator.\n");
            fprintf(filein,";Specification for this file is:\n");
            fprintf(filein,";%d values with a %d bit width and base
%d.\n",rd,rw,rad);
            fprintf(filein,"radix = %d;\n",rad);
            fprintf(filein,"patern =\n");
            break;
      case 4:
            fprintf(filein,";COE FILE\n");
            fprintf(filein,";\n");
            fprintf(filein,";The data that follows comprises the
coefficients of a interleaver.\n");
            fprintf(filein,";Specification for this file is:\n");
            fprintf(filein,";%d values with a %d bit width and base
%d.\n",rd,rw,rad);
            fprintf(filein,"radix = %d;\n",rad);
            fprintf(filein,"branch_length_vector =\n");
            break;
      case 5:
            fprintf(filein,";COE FILE\n");
            fprintf(filein,";\n");
            fprintf(filein,";The data that follows comprises the contents
of a memory.\n");
            fprintf(filein,";Specification for this file is:\n");
            fprintf(filein,";%d values with a %d bit width and base
%d.\n",rd,rw,rad);
            fprintf(filein,"memory_initialization_radix = %d;\n",rad);
            fprintf(filein,"memdata =\n");
            break;
      }

      return;
}

void makerandlist(int rd, int rw, unsigned int* list){

      int i;

      seedMT((unsigned)time(NULL));

      for (i=0; i<rd; i++) {
            list[i] = FILTER(rw)&randomMT();
      }

      return;
}

void workinput(int rd,int rw,int rad, unsigned int* list){

      int i;
```

```
	unsigned int curvalue;
	char curvaluestr[BIN_STR_MAXSIZE];


	printf("Enter your values one by one and press enter after
each\n");
	for (i=1;i<rd+1;i++){
		printf("Value index %d:\t",i);
		scanf("%u",&list[i-1]);

		if(list[i-1]>=(pow(2.0,(long)rw))){
			//crop bit info above selected ramwidth
			list[i-1] = FILTER(rw)&list[i-1];

		}

		curvalue=list[i-1];

		switch (rad){
		case 2:
			dec2binconv(curvalue,rw,curvaluestr);
			printf("Your value will be printed as
%s\n",curvaluestr);
			break;
		case 10:
			printf("Your value will be printed as %u\n",curvalue);
			break;
		case 16:
			printf("Your value will be printed as
%0*x\n",(int)ceil((double)rw/4),curvalue);
			break;
		}
	}
}

void printdata(FILE* fileout,int rd,int rw,int radix, unsigned int*
list){

	int i;
	char curvaluestr[BIN_STR_MAXSIZE];

	for(i=0;i<rd;i++) {
		switch (radix) {
		case 2:
			dec2binconv(list[i],rw,curvaluestr);
			if(i!=rd-1){
				fprintf(fileout,"%s,\n",curvaluestr);
			}
			else{
				fprintf(fileout,"%s;",curvaluestr);
			}
```

```
//    printf("Value %s printed to file\n",curvaluestr);
                  break;
            case 10:
                  if(i!=rd-1){
                        fprintf(fileout,"%u,\n",list[i]);
                  }
                  else{
                        fprintf(fileout,"%u;",list[i]);
                  }
//    printf("Value %u printed to file\n",list[i]);
                  break;
            case 16:
                  if(i!=rd-1){

      fprintf(fileout,"%0*X,\n",(int)ceil((double)rw/BITS_PER_HEXA_CHAR),
list[i]);
                  }
                  else{

      fprintf(fileout,"%0*X;",(int)ceil((double)rw/BITS_PER_HEXA_CHAR),li
st[i]);
                  }
//    printf("Value              %0*X              printed              to
file\n",(int)ceil((double)rw/BITS_PER_HEXA_CHAR),list[i]);
                  break;
            }
      }
      printf("Completes print data successfully\n"); //TESTING
      return;

}

void dec2binconv(unsigned int value,int ramwidth,char *binstr){

      int  k, n = 0;
      unsigned int  val, r;
      char tempstr[BIN_STR_MAXSIZE];                  //16 bit values plus
termminating \0

      val=value;
      k=0;
      n=0;

      while (val>0) {
            r = val%2;                    //get remainder
            val = val / 2;                //setup next iteration
            tempstr[k++] = r + '0'; //conversion    of    remainder    to
apropriate char
      }
```

```
      while (k<(ramwidth)){        //  fill  remaining  bits  with  '0'  in
accordance with ramwidth
            tempstr[k++]='0';
      }

      while (k >= 0){                    // reverse string
            binstr[n++] = tempstr[--k];
      }
      binstr[n-1] = 0;
}
```

ANEXO E

Hardware USB-EPP implementado na Nexys 2 e Atlys

```
--
-- Copyright (C) 2011 Chris McClelland
--
-- This program is free software: you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as published by
-- the Free Software Foundation, either version 3 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program.  If not, see <http://www.gnu.org/licenses/>.
--
--
--
-- Heavily Edited to allow 32 bit data and 16 bit addresses by Pedro
Soares
-- Universidade de Aveiro - DETI
-- pedrosoares@ua.pt
-- 27987
--
--
--

library ieee;

use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--use ieee.std_logic_signed.all;
use IEEE.numeric_std.all;

entity USB_EPP_FSM is
      generic(
             AddrSize: positive);
      port(
            clk                     : in        std_logic;
                              --clk signal
            reset                   : in        std_logic;
                              --reset signal
            eppDataBus        : inout     std_logic_vector(7 downto 0);
      --epp data bus
            eppAddrStrobe     : in        std_logic;
                        --epp address strobe
            eppDataStrobe     : in        std_logic;
                        --epp data strobe
            eppWrite                : in        std_logic;
                              --epp write signal
            eppWait           : out       std_logic;
                        --epp wait signal
```

```
            DataWrEn                        : out       std_logic_vector(0
downto 0);        --data write enable
            Address16bit            : out       std_logic_vector(AddrSize-
1 downto 0);      --(up to) 16 bit address received
            DataIn32bit             : out       std_logic_vector(31 downto
0);   --32 bit data received
            DataOut32bit            : in        std_logic_vector(31 downto
0)          --32 bit data requested
                  );
end USB_EPP_FSM;

architecture Behavioural of USB_EPP_FSM is
      type State is (
            STATE_IDLE,

            STATE_ADDR_WRITE_EXEC,
            STATE_ADDR_WRITE_ACK,

            STATE_DATA_WRITE_EXEC,
            STATE_DATA_WRITE_ACK,

            STATE_DATA_READ_EXEC,
            STATE_DATA_READ_ACK
      );

      -- State and next-state
      signal CurrentState, NextState : State;

      -- Synchronised versions of asynchronous inputs
      signal sAddrStrobe      : std_logic;
      signal sDataStrobe   : std_logic;
      signal sWrite        : std_logic;

      -- Data to be output
      signal DataOutput : std_logic_vector(7 downto 0);

      -- Registers
      signal CurrentWait, NextWait              : std_logic;
      signal CurrentOutData, NextOutData        :       std_logic_vector(7
downto 0);
      signal CurrentWrEn, NextWrEn                    : std_logic_vector(0
downto 0);
      signal CurrentUpper, NextUpper                  : std_logic_vector(2
downto 0) := "000";
      signal CurrentAddr16b, NextAddr16b        :
std_logic_vector(AddrSize-1 downto 0);
      signal CurrentDataIn32b, NextDataIn32b    :      std_logic_vector(31
downto 0);

begin

      Address16bit <= CurrentAddr16b;
      DataIn32bit <= CurrentDataIn32b;
      DataWrEn <= CurrentWrEn;

      -- Drive the outputs
      eppWait <= CurrentWait;

      -- EPP operation
```

```
eppDataBus <= DataOutput when ( eppWrite = '1' ) else "ZZZZZZZZ";

DataOutput <= CurrentOutData;

-- Infer registers
process(clk, reset)
begin
      if rising_edge(clk) then
            if ( reset = '1' ) then
                  CurrentState      <= STATE_IDLE;
                  CurrentAddr16b    <= (others => '0');
                  CurrentOutData    <= (others => '0');
                  CurrentDataIn32b  <= x"00000000";
                  CurrentWait       <= '0';
                  sAddrStrobe             <= '1';
                  sDataStrobe             <= '1';
                  sWrite                        <= '1';
                  CurrentWrEn             <= "0";
                  CurrentUpper            <= "000";
            else
                  CurrentState      <= NextState;
                  CurrentAddr16b    <= NextAddr16b;
                  CurrentOutData    <= NextOutData;
                  CurrentDataIn32b  <= NextDataIn32b;
                  CurrentWait       <= NextWait;
                  sAddrStrobe             <= eppAddrStrobe;
                  sDataStrobe             <= eppDataStrobe;
                  sWrite                        <= eppWrite;
                  CurrentWrEn             <= NextWrEn;
                  CurrentUpper            <= NextUpper;
            end if;
      end if;
end process;

-- Next state logic
process( eppDataBus, CurrentState, CurrentAddr16b, sAddrStrobe,
sDataStrobe, sWrite, CurrentOutData, DataOut32bit,
CurrentWrEn, CurrentUpper, CurrentDataIn32b)

begin
      NextWait       <= '0';
      NextState      <= STATE_IDLE;
      NextAddr16b    <= CurrentAddr16b;
      NextOutData    <= CurrentOutData;
      NextWrEn              <= CurrentWrEn;
      NextUpper        <= CurrentUpper;
      NextDataIn32b    <= CurrentDataIn32b;

      case CurrentState is
            when STATE_IDLE =>
                  if ( sAddrStrobe = '0' ) then
                        -- Address can only be written, not read
                        if ( sWrite = '0' ) then
                              NextState <= STATE_ADDR_WRITE_EXEC;
                        end if;
                  elsif ( sDataStrobe = '0' ) then
                        -- Register read or write
                        if ( sWrite = '0' ) then
                              NextState <= STATE_DATA_WRITE_EXEC;
```

```
                              else
                                    NextState <= STATE_DATA_READ_EXEC;
                              end if;
                        end if;

                  -- Write address register
                  when STATE_ADDR_WRITE_EXEC =>
                        if (AddrSize > 8) then
                              if (CurrentUpper = "000") then
                                    NextAddr16b(AddrSize-1  downto  8)  <=
eppDataBus(AddrSize-1-8 downto 0);
                              elsif (CurrentUpper = "001") then
                                    NextAddr16b(7     downto     0)     <=
eppDataBus;
                              else
                                    null;
                              end if;
                        else
                              if (CurrentUpper = "001") then
                                    NextAddr16b(AddrSize-1  downto  0)  <=
eppDataBus(AddrSize-1 downto 0);
                              else
                                    null;
                              end if;
                        end if;
                        NextState   <= STATE_ADDR_WRITE_ACK;
                        NextWait     <= '0';
                  when STATE_ADDR_WRITE_ACK =>
                        if ( sAddrStrobe = '0' ) then
                              NextState <= STATE_ADDR_WRITE_ACK;
                              NextWait   <= '1';
                        else
                              NextState <= STATE_IDLE;
                              NextWait   <= '0';
                        end if;

                  -- Write data register
                  when STATE_DATA_WRITE_EXEC =>
                        if (CurrentUpper = "000") then
                              NextDataIn32b(31 downto 24) <= eppDataBus;
                        elsif (CurrentUpper = "001") then
                              NextDataIn32b(23 downto 16) <= eppDataBus;
                        elsif (CurrentUpper = "010") then
                              NextDataIn32b(15 downto 8) <= eppDataBus;
                        elsif (CurrentUpper = "011") then
                              NextDataIn32b(7 downto 0) <= eppDataBus;
                        else
                              null;
                        end if;
                        NextState <= STATE_DATA_WRITE_ACK;
                        NextWait   <= '1';
                  when STATE_DATA_WRITE_ACK =>
                        if ( sDataStrobe = '0' ) then
                              if (CurrentUpper = "011") then
                                    NextWrEn <= "1";
                              end if;
                              NextState <= STATE_DATA_WRITE_ACK;
                              NextWait   <= '1';
                        else
```

```
                              if (CurrentUpper < "011") then
                                    NextUpper <= CurrentUpper + 1;
                              else
                                    NextUpper <= "000";
                              end if;
                              NextWrEn <= "0";
                              NextState <= STATE_IDLE;
                              NextWait   <= '0';
                        end if;

                  -- Read data register
                  when STATE_DATA_READ_EXEC =>
                        if (CurrentUpper = "001") then
                              NextOutData <= DataOut32bit(31 downto 24);
                        elsif (CurrentUpper = "010") then
                              NextOutData <= DataOut32bit(23 downto 16);
                        elsif (CurrentUpper = "011") then
                              NextOutData <= DataOut32bit(15 downto 8);
                        elsif (CurrentUpper = "100") then
                              NextOutData <= DataOut32bit(7 downto 0);
                        else
                              null;
                        end if;
                        NextWait    <= '1';
                        NextState  <= STATE_DATA_READ_ACK;
                  when STATE_DATA_READ_ACK =>
                        if ( sDataStrobe = '0' ) then
                              NextState <= STATE_DATA_READ_ACK;
                              NextWait   <= '1';
                        else
                              if (CurrentUpper < "100") then
                                    NextUpper <= CurrentUpper + 1;
                              else
                                    NextUpper <= "000";
                              end if;
                              NextState <= STATE_IDLE;
                              NextWait   <= '0';
                        end if;

                  -- Some unknown state
                  when others =>
                        NextState <= STATE_IDLE;
            end case;
      end process;

end Behavioural;
```

ANEXO F

Algoritmo de envio por USB-EPP de uma palavra de 32 bits para a blockRAM

```
void Send4BFPGA()
{
      unsigned int memind, memind_t;
      unsigned int memval, memval_t;

      /* Send 32 bits to FPGA using DoPutReg() function */
      memind = atoi(szRegister);
      memval = atoi(szByte);

      memind_t = (0x0000FF00&memind)>>8;
      _itoa(memind_t, szRegister, 10);
      memval_t = (0xFF000000&memval)>>24;
      _itoa(memval_t, szByte, 10);
      DoPutReg();
      memind_t = (0x000000FF&memind);
      _itoa(memind_t, szRegister, 10);
      memval_t = (0x00FF0000&memval)>>16;
      _itoa(memval_t, szByte, 10);
      DoPutReg();
      memval_t = (0x0000FF00&memval)>>8;
      _itoa(memval_t, szByte, 10);
      DoPutReg();
      memval_t = (0x000000FF&memval);
      _itoa(memval_t, szByte, 10);
      DoPutReg();

      printf("Complete. Memory index %d set with value %d.\n", memind
,memval);
 }
```

ANEXO G

Algoritmo de leitura por USB-EPP de uma palavra de 32 bits presente na blockRAM

```
void Get4BFPGA(){

	unsigned int memind, memind_t;
	unsigned int memval;
	unsigned int temp1,temp2, temp3, temp4;

	/* Receive 32 bits from FPGA using DoGetReg() function */
	memind = atoi(szRegister);
	memind_t = (0x0000FF00&memind)>>8;
	_itoa(memind_t, szRegister, 10);
	DoGetReg(); //1
		memind_t = (0x000000FF&memind);
	_itoa(memind_t, szRegister, 10);
	DoGetReg(); //2
	temp1 = idData;
	DoGetReg(); //3
	temp2 = idData;
	DoGetReg(); //4
	temp3 = idData;
	DoGetReg(); //5
	temp4 = idData;

	memval                                                                  =
(((0x00FF&temp1)<<24)+((0x00FF&temp2)<<16)+((0x00FF&temp3)<<8)+(0x00FF&te
mp4));

	printf("Complete.  Memory  index  %d  reported  value  %d.\n",memind,
memval);

}
```

ANEXO H

Configuração de UART e serviço de atendimento à interrupção genérico para os diversos projetos

```
/* Initialize RS232 - Set baudrate and number of stop bits */
      XUartNs550_SetBaud(XPAR_RS232_UART_1_BASEADDR,
XPAR_RS232_UART_1_CLOCK_FREQ_HZ, 115200);
      XUartNs550_SetLineControlReg(XPAR_RS232_UART_1_BASEADDR,
XUN_LCR_8_DATA_BITS);

      XCACHE_ENABLE_ICACHE();
      XCACHE_ENABLE_DCACHE();

      // Introduce project
      intro();

      //initialize interrupt controller
      Status = XIntc_Initialize (&Intc, XPAR_XPS_INTC_0_DEVICE_ID);
    if (Status != XST_SUCCESS) xil_printf ("\r\nInterrupt controller
initialization failure");
    else xil_printf("\r\nInterrupt controller initialized");

    //register usb_epp_interrupt_handler() for the system
    Status = XIntc_Connect (&Intc,
XPAR_XPS_INTC_0_DIGILENT_USB_EPP_IRQ_EPP_INTR, (XInterruptHandler)
usb_epp_interrupt_handler, (void *)XPAR_DIGILENT_USB_EPP_BASEADDR);

    if (Status != XST_SUCCESS) xil_printf ("\r\nRegistering USB_EPP
Interrupt Failed");
    else xil_printf("\r\nUSB_EPP Interrupt registered");

    //enable interrupt controller
    XIntc_Enable  (&Intc, XPAR_XPS_INTC_0_DIGILENT_USB_EPP_IRQ_EPP_INTR);
    xil_printf("\r\nInterrupt controller enabled");

    //start the interrupt controller in real mode
    Status = XIntc_Start(&Intc, XIN_REAL_MODE);
    if (Status != XST_SUCCESS) xil_printf ("\r\nInterupt controller
starting failed");
    else xil_printf("\r\nInterrupt controller started");

    //enable interrupts for the processor
    microblaze_enable_interrupts();
```

ANEXO I

Biblioteca de Funções implementadas para controlo da Flash presente na Nexys 2

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

// These functions are for Nexys 2's FLASH - a Numonyx J3 75 128Mbit

#define REVERSEBITS(b) (BitReverseTable[b])

#define FLASHBASEADDRESS      XPAR_INTEL_FLASH_MEM0_BASEADDR
#define FLASHHIGHADDRESS      XPAR_INTEL_FLASH_MEM0_HIGHADDR
#define     UARTBASEADDRESS         XPAR_RS232_PORT_BASEADDR
#define MAX_ADDR              0x00FFFFFF
#define FILTER0                     0xFF000000
#define FILTER1                     0x00FF0000
#define FILTER2                     0x0000FF00
#define FILTER3                     0x000000FF
#define FILTERUP              0xFF00
#define FILTERLOW             0x00FF
#define FILTERSR7             0x80
#define FLASHSECTORS          128
#define SECTORSIZE                  0x20000

//    Registers
#define ProgramEnhancedConfigurationRegister   0x0060
#define ProgramEnhancedConfigurationRegister2  0x0004
#define ProgramOTPRegister                                  0x00C0
#define ClearStatusRegister                           0x0050
#define ProgramSTSConfigurationRegister        0x00B8

//    Read Modes
#define ReadArray                                                 0x00FF
#define ReadStatusRegister                                  0x0070
#define ReadIdentifierCodes                                 0x0090
#define CFIQuery                                                  0x0098

//    Program and Erase
#define WordProgram                                         0x0040
#define ByteProgram                                               0x0010
#define BufferedProgram                                     0x00E8
#define BufferedProgram2                                    0x00D0
#define BlockErase                                                0x0020
#define BlockErase2                                         0x00D0
#define ProgramEraseSuspend                           0x00B0
#define ProgramEraseResume                            0x00D0
```

```
//    Security
#define LockBlock                                                    0x0060
#define LockBlock2                                                   0x0001
#define UnlockBlock                                            0x0060
#define UnlockBlock2                                           0x00D0

#include <stdio.h>
#include <stdlib.h>
#include "xparameters.h"
#include "xenv_standalone.h"
#include "xio.h"
#include "xuartns550_l.h"

static Xuint8 BitReverseTable[256];

void write_to_flash(Xuint32 addr, Xuint16 writeword) {

      Xuint16 tempval;
      Xuint8 uppertemp, lowertemp;

      XIo_Out8(FLASHBASEADDRESS+addr, REVERSEBITS(ByteProgram)); // issue
write word command
      uppertemp = (Xuint8)((writeword&FILTERUP)>>8);
      //xil_printf("\r\nUpper byte to write was 0x%02X\r\n", uppertemp);
      uppertemp = REVERSEBITS(uppertemp);
      uppertemp = (Xuint16) uppertemp;
      //xil_printf("\r\nUpper  byte  to  write  reversed  was  0x%02X\r\n",
uppertemp);
      lowertemp = writeword&FILTERLOW;
      //xil_printf("\r\nLower byte to write was 0x%02X\r\n", lowertemp);
      lowertemp = REVERSEBITS(lowertemp);
      //xil_printf("\r\nLower  byte  to  write  reversed  was  0x%02X\r\n",
lowertemp);
      tempval = (uppertemp<<8)+lowertemp;
      //xil_printf("\r\n Original word is 0x%04X and reversed word is
0x%04X\r\n", writeword, tempval);
      XIo_Out16(FLASHBASEADDRESS+addr, tempval);
      do {
            XIo_Out8(FLASHBASEADDRESS, REVERSEBITS(ReadStatusRegister));
            tempval=XIo_In8(FLASHBASEADDRESS);
            tempval=REVERSEBITS(tempval);
            //xil_printf("\r\nReceived SR word is 0x%02X\r\n", tempval);
      }while ((tempval & FILTERSR7)!=FILTERSR7);
}

void read_from_flash(Xuint32 addr, Xuint16 *readword) {
            //NOW WORKING

      Xuint16 tempval;
      Xuint8 uppertemp, lowertemp;

      XIo_Out8(FLASHBASEADDRESS, REVERSEBITS(ReadArray));        //  Read
word command
      tempval=XIo_In16(FLASHBASEADDRESS+addr);             //   Get   word
from address
      //xil_printf("\r\nValue read was 0x%04X\r\n", tempval);
      uppertemp = (Xuint8)((tempval&FILTERUP)>>8);
      //xil_printf("\r\nUpper byte read was 0x%02X\r\n", uppertemp);
      uppertemp = REVERSEBITS(uppertemp);
```

```
	//xil_printf("\r\nUpper byte reversed was 0x%02X\r\n", uppertemp);
	lowertemp = tempval&FILTERLOW;
	//xil_printf("\r\nLower byte read was 0x%02X\r\n", lowertemp);
	lowertemp = REVERSEBITS(lowertemp);
	//xil_printf("\r\nLower byte reversed was 0x%02X\r\n", lowertemp);
	*readword = (uppertemp<<8) + lowertemp;
}

void erase_flash_sector(Xuint32 addr) {

	Xuint8 tempval;

	XIo_Out8(FLASHBASEADDRESS+addr, REVERSEBITS(BlockErase));  // issue
erase sector command 1
	XIo_Out8(FLASHBASEADDRESS+addr, REVERSEBITS(BlockErase2)); // issue
erase sector command 2

	do {
		XIo_Out8(FLASHBASEADDRESS, REVERSEBITS(ReadStatusRegister));
		tempval=XIo_In8(FLASHBASEADDRESS);
		tempval=REVERSEBITS(tempval);
		//xil_printf("\r\nReceived SR word is 0x%02X\r\n", tempval);
	}while ((tempval & FILTERSR7)!=FILTERSR7);
	xil_printf("\r\nSector    nr.    %d    of    128    erased.\r\n",
(addr/SECTORSIZE)+1);

}

void erase_flash(void){

	int i;

	xil_printf("\r\nNow erasing flash one sector at a time\r\n");
	for (i=0;i<FLASHSECTORS;i++) {
		erase_flash_sector(i*SECTORSIZE);
	}
	xil_printf("\r\nFlash Memory Successfully erased\r\n");

}

void read_status(Xuint8 *regstat) {

	XIo_Out8(FLASHBASEADDRESS, REVERSEBITS(ReadStatusRegister));
	*regstat = REVERSEBITS(XIo_In8(FLASHBASEADDRESS));

}

void read_id(Xuint16 *regid) {

	XIo_Out16(FLASHBASEADDRESS, REVERSEBITS(ReadIdentifierCodes));
	*regid = REVERSEBITS(XIo_In16(FLASHBASEADDRESS+0x1));

}


/*http://www.fileformat.info/mirror/egff/ch06_04.htm*/
static Xuint8 BitReverseTable[256] = {
0x00, 0x80, 0x40, 0xc0, 0x20, 0xa0, 0x60, 0xe0,
0x10, 0x90, 0x50, 0xd0, 0x30, 0xb0, 0x70, 0xf0,
```

```
0x08, 0x88, 0x48, 0xc8, 0x28, 0xa8, 0x68, 0xe8,
0x18, 0x98, 0x58, 0xd8, 0x38, 0xb8, 0x78, 0xf8,
0x04, 0x84, 0x44, 0xc4, 0x24, 0xa4, 0x64, 0xe4,
0x14, 0x94, 0x54, 0xd4, 0x34, 0xb4, 0x74, 0xf4,
0x0c, 0x8c, 0x4c, 0xcc, 0x2c, 0xac, 0x6c, 0xec,
0x1c, 0x9c, 0x5c, 0xdc, 0x3c, 0xbc, 0x7c, 0xfc,
0x02, 0x82, 0x42, 0xc2, 0x22, 0xa2, 0x62, 0xe2,
0x12, 0x92, 0x52, 0xd2, 0x32, 0xb2, 0x72, 0xf2,
0x0a, 0x8a, 0x4a, 0xca, 0x2a, 0xaa, 0x6a, 0xea,
0x1a, 0x9a, 0x5a, 0xda, 0x3a, 0xba, 0x7a, 0xfa,
0x06, 0x86, 0x46, 0xc6, 0x26, 0xa6, 0x66, 0xe6,
0x16, 0x96, 0x56, 0xd6, 0x36, 0xb6, 0x76, 0xf6,
0x0e, 0x8e, 0x4e, 0xce, 0x2e, 0xae, 0x6e, 0xee,
0x1e, 0x9e, 0x5e, 0xde, 0x3e, 0xbe, 0x7e, 0xfe,
0x01, 0x81, 0x41, 0xc1, 0x21, 0xa1, 0x61, 0xe1,
0x11, 0x91, 0x51, 0xd1, 0x31, 0xb1, 0x71, 0xf1,
0x09, 0x89, 0x49, 0xc9, 0x29, 0xa9, 0x69, 0xe9,
0x19, 0x99, 0x59, 0xd9, 0x39, 0xb9, 0x79, 0xf9,
0x05, 0x85, 0x45, 0xc5, 0x25, 0xa5, 0x65, 0xe5,
0x15, 0x95, 0x55, 0xd5, 0x35, 0xb5, 0x75, 0xf5,
0x0d, 0x8d, 0x4d, 0xcd, 0x2d, 0xad, 0x6d, 0xed,
0x1d, 0x9d, 0x5d, 0xdd, 0x3d, 0xbd, 0x7d, 0xfd,
0x03, 0x83, 0x43, 0xc3, 0x23, 0xa3, 0x63, 0xe3,
0x13, 0x93, 0x53, 0xd3, 0x33, 0xb3, 0x73, 0xf3,
0x0b, 0x8b, 0x4b, 0xcb, 0x2b, 0xab, 0x6b, 0xeb,
0x1b, 0x9b, 0x5b, 0xdb, 0x3b, 0xbb, 0x7b, 0xfb,
0x07, 0x87, 0x47, 0xc7, 0x27, 0xa7, 0x67, 0xe7,
0x17, 0x97, 0x57, 0xd7, 0x37, 0xb7, 0x77, 0xf7,
0x0f, 0x8f, 0x4f, 0xcf, 0x2f, 0xaf, 0x6f, 0xef,
0x1f, 0x9f, 0x5f, 0xdf, 0x3f, 0xbf, 0x7f, 0xff
};
```

ANEXO J

Projeto de controlo da Flash presente na Nexys 2

```
/*
 *
 * Heavily edited by
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

#define FLASHBASEADDRESS      XPAR_INTEL_FLASH_MEM0_BASEADDR
#define FLASHHIGHADDRESS      XPAR_INTEL_FLASH_MEM0_HIGHADDR
#define     UARTBASEADDRESS         XPAR_RS232_PORT_BASEADDR
#define MAX_ADDR              0x00FFFFFF
#define FILTER0                     0xFF000000
#define FILTER1                     0x00FF0000
#define FILTER2                     0x0000FF00
#define FILTER3                     0x000000FF
#define     TESTWORD16              0xA5C3
#define EMPTYCELLS                  0xFFFF

#include <stdio.h>
#include <stdlib.h>
#include "xparameters.h"
#include "xenv_standalone.h"
#include "xbasic_types.h"
#include "xio.h"
#include "xuartns550_l.h"

// Function prototypes
u8 operchoice(void);
Xuint8 simpleHex2Dec(char hexin);
int wordRead(Xuint16 *wordtoread);
void read_from_flash(Xuint32 addr, Xuint16 *readword);
void write_to_flash(Xuint32 addr, Xuint16 writeword);
void erase_flash_sector(Xuint32 addr);
void read_status(Xuint8 *regstat);
void read_id(Xuint8 *regid);
void printopts(void);
void erase_flash(void);
int dataCountEntry(Xuint32 *count);
int addresSselect(Xuint32 *addr);
Xuint8 simpleHex2Dec(char hexin);
int byteRead(Xuint16 *bytetoread);
void intro(void);

int main() { //former main

      u8 oper;
      Xuint32     address, count, curaddress;
      Xuint16 recbyte;// idword;
      Xuint8 statword;
      int ret, i, err;
```

```
	/* Initialize RS232 - Set baudrate and number of stop bits */
	XUartNs550_SetBaud(UARTBASEADDRESS,  XPAR_RS232_PORT_CLOCK_FREQ_HZ,
115200);
	XUartNs550_SetLineControlReg(UARTBASEADDRESS, XUN_LCR_8_DATA_BITS);

	intro();

	while (1) {
		printopts();
		oper = operchoice();
		switch (oper) {

		case 'a':
			xil_printf("Now erasing Flash memory to ensure proper
write operations.\r\n");
			xil_printf("Please hold...\r\n");
			erase_flash();
			xil_printf("Erasure  procedure  complete,  now  testing
Flash memory.\r\n");
			for (i=0;i<(MAX_ADDR+1);i=i+2){
				write_to_flash(i, TESTWORD16);
			}
			err=0;
			for (i=0;i<(MAX_ADDR+1);i=i+2) {
				read_from_flash(i, &recbyte);
				if (recbyte != TESTWORD16) {
					xil_printf("Difference  found  in  address
0x%08X, expected 0xA5C3 and read 0x%04X\r\n", i, recbyte);
					err++;
				}
			}
			if (err != 0) xil_printf("Memory test failed, %d 16 bit
words differed from test word\r\n", err);
			else	xil_printf("Memory	test	completed
successfully\r\n");

			break;



		case 'd':
			ret = addresSselect(&address);
			if (ret!=0x00) {
				xil_printf("Address selection failed\r\n");
				goto dexit;
			}
			//data count read
			ret = dataCountEntry(&count);
			if (ret!=0x00) {
				xil_printf("Data amount selection failed\r\n");
				goto dexit;
			}
			if (address > MAX_ADDR) {
				xil_printf("Operation  would  address  over  the
Flash size\r\n");
				xil_printf("Thus it will now break\r\n");
				goto dexit;
			}
```

```
                if ((address + count) > MAX_ADDR) {
                      xil_printf("Operation partly addresses over the
Flash size\r\n");
                      xil_printf("Thus the amount of data required
cannot be fulfilled\r\n");
                      count = ((MAX_ADDR + 1) - address);
                      xil_printf("For this opp, only 0x%08X bytes will
be read\r\n", count);
                }
                for (i=0;i<count;i=i+2) {
                      curaddress = address + i;
                      if ((i%32) == 0) xil_printf("\r\nFlash Address
0x%08X ",curaddress);
                      read_from_flash(address+i, &recbyte);
                      xil_printf("0x%04X ", recbyte);
                }
                xil_printf("\r\n\r\nData read successfully.\r\n");
dexit:          break;


          case 'c':
                ret = addresSselect(&address);
                if (ret!=0x00) {
                      xil_printf("Address selection failed\r\n");
                      goto cexit;
                }
                //data count read
                ret = dataCountEntry(&count);
                if (ret!=0x00) {
                      xil_printf("Data amount selection failed\r\n");
                      goto cexit;
                }
                if (address > MAX_ADDR) {
                      xil_printf("Operation would address over the
Flash size\r\n");
                      xil_printf("Thus it will now break\r\n");
                      goto cexit;
                }
                if ((address + count) > MAX_ADDR) {
                      xil_printf("Operation partly addresses over the
Flash size\r\n");
                      xil_printf("Thus the amount of data required
cannot be fulfilled\r\n");
                      count = ((MAX_ADDR + 1) - address);
                      xil_printf("For this opp, only 0x%08X bytes will
be read\r\n", count);
                }
                for (i=0;i<count;i=i+2){
                      xil_printf("Enter your data word now (16 bit, 4
hex chars)\r\n");
                      ret = byteRead(&recbyte);
                      if (ret != 0){
                            xil_printf("Data value reading failed,
operation has failed.\r\n");
                            goto cexit;
                      }
                      write_to_flash(address+i, recbyte);
                      xil_printf("\r\nByte printed was 0x%04X\r\n",
recbyte);
```

```
                }
                xil_printf("\r\n\r\nData written successfully.\r\n");
cexit:          break;


          case 'b':
                ret = addresSselect(&address);
                if (ret!=0x00) {
                      xil_printf("Address selection failed\r\n");
                      goto bexit;
                }
                erase_flash_sector(address);
bexit:          break;

          case 'h':
                read_status(&statword);
                xil_printf("\r\nStatus byte read was 0x%02X\r\n",
statword);
                break;

          /*case 'e':
                read_id(&idword);
                xil_printf("\r\nID read was 0x%02X\r\n", statword);
                break;*/

          case 'g':
                erase_flash();
                break;

          case 'f':
                xil_printf("Now checking Flash memory to ensure all
cells are empty.\r\n");
                xil_printf("Please hold...\r\n");
                err=0;
                for (i=0;i<(MAX_ADDR+1);i=i+2) {
                      read_from_flash(i, &recbyte);
                      if (recbyte != EMPTYCELLS) {
                            xil_printf("Difference found in address
0x%08X, expected 0xFFFF and read 0x%04X\r\n", i, recbyte);
                            err++;
                      }
                }
                if (err != 0) xil_printf("Flash is not empty, %d 16 bit
words differed from empty cells word 0xFFFF.\r\n", err);
                else xil_printf("Flash is empty.\n");
                break;

          case 'e':
                rand();                                    //workaround
to avoid sequence repetition
                srand(5);
                ret = addresSselect(&address);
                if (ret!=0x00) {
                      xil_printf("Address selection failed\r\n");
                      goto eexit;
                }
                //data count read
                ret = dataCountEntry(&count);
                if (ret!=0x00) {
```

```
                        xil_printf("Data amount selection failed\r\n");
                        goto eexit;
                  }
                  if (address > MAX_ADDR) {
                        xil_printf("Operation  would  address  over  the
Flash size\r\n");
                        xil_printf("Thus it will now break\r\n");
                        goto eexit;
                  }
                  if ((address + count) > MAX_ADDR) {
                        xil_printf("Operation  partly  addresses  over  the
Flash size\r\n");
                        xil_printf("Thus  the  amount  of  data  required
cannot be fulfilled\r\n");
                        count = ((MAX_ADDR + 1) - address);
                        xil_printf("For this opp, only 0x%08X bytes will
be read\r\n", count);
                  }
                  for (i=0;i<count;i=i+2){
                        write_to_flash(address+i, (rand()&EMPTYCELLS));
                        //xil_printf("\r\nByte  printed  was  0x%04X\r\n",
recbyte);
                  }
                  xil_printf("\r\n\r\nData written successfully.\r\n");
eexit:            break;

            default:
                  xil_printf("\r\nBad oper chosen, will try again\r\n");
                  break;
            }
      }
      return 0;
}

u8 operchoice(void) {
      u8 operchoice;

      operchoice = XUartNs550_RecvByte(UARTBASEADDRESS);
      xil_printf("\r\nThe Selected Operation was: %c) \r\n",operchoice);
      return operchoice;
}

int wordRead(Xuint16 *wordtoread){
      u8 buffer[4];
      int ret0, ret1, ret2, ret3;

      xil_printf("Enter your word value now\r\n");
      buffer[0]=XUartNs550_RecvByte(UARTBASEADDRESS);
      xil_printf(" %c", buffer[0]);
      buffer[1]=XUartNs550_RecvByte(UARTBASEADDRESS);
      xil_printf("%c",buffer[1]);
      buffer[2]=XUartNs550_RecvByte(UARTBASEADDRESS);
      xil_printf("%c",buffer[2]);
      buffer[3]=XUartNs550_RecvByte(UARTBASEADDRESS);
      xil_printf("%c",buffer[3]);
      ret0=simpleHex2Dec(buffer[0]);
      ret1=simpleHex2Dec(buffer[1]);
      ret2=simpleHex2Dec(buffer[2]);
      ret3=simpleHex2Dec(buffer[3]);
```

```
	if ((ret0<0)|(ret1<0)|(ret2<0)|(ret3<0)){
		xil_printf("You've entered a char that is not an hex value!\r\n");
		return -1;
	}
	else *wordtoread=((ret0<<12)+(ret1<<8)+(ret2<<4)+ret3);
	return 0;
}

Xuint8 simpleHex2Dec(char hexin){
	if (hexin=='0')return 0x0;
	if (hexin=='1')return 0x1;
	if (hexin=='2')return 0x2;
	if (hexin=='3')return 0x3;
	if (hexin=='4')return 0x4;
	if (hexin=='5')return 0x5;
	if (hexin=='6')return 0x6;
	if (hexin=='7')return 0x7;
	if (hexin=='8')return 0x8;
	if (hexin=='9')return 0x9;
	if (hexin=='a')return 0xa;
	if (hexin=='b')return 0xb;
	if (hexin=='c')return 0xc;
	if (hexin=='d')return 0xd;
	if (hexin=='e')return 0xe;
	if (hexin=='f')return 0xf;
	if (hexin=='A')return 0xa;
	if (hexin=='B')return 0xb;
	if (hexin=='C')return 0xc;
	if (hexin=='D')return 0xd;
	if (hexin=='E')return 0xe;
	if (hexin=='F')return 0xf;
	else return 0xFF;
}

void printopts(void) {
	xil_printf("\r\n\r\nEnter your Flash operation now\r\n");
	xil_printf("a)  Write/Read  Test  Entire  Memory  (destructive test)\r\n");
	xil_printf("b) Erase Sector from Flash\r\n");
	xil_printf("c) Write Data to Flash\r\n");
	xil_printf("d) Read Data from Flash\r\n");
	xil_printf("e) Generate And Write Random Words\r\n");
	xil_printf("f) Blank Check Entire Memory\r\n");
	xil_printf("g) Erase Entire Flash\r\n");
	xil_printf("h) Read Status Register\r\n");
	/*xil_printf("e) read flash id code\r\n");*/
	xil_printf(":");
}

int dataCountEntry(Xuint32 *count){
	u8 buffer[8];
	int i=0, ret;
	Xuint32 temp=0x00000000;

	xil_printf("\r\nChoose the amount of data you wish to use in this op.\r\n");
	xil_printf("Terminate your option filling in the 8 hex chars that represent that value.\r\n");
```

```
	for (i=0;i<8;i++){
		buffer[i] = XUartNs550_RecvByte(UARTBASEADDRESS);
		xil_printf("%c",buffer[i]);
	}
	i=0;
	while ((ret!=0xFF)&(i<8)){
		ret=simpleHex2Dec(buffer[i]);
		temp = (temp<<(4)) + ret;
		i+=1;
	}
	if (ret==0xFF){
		xil_printf("Bad choice of data amount, try again!\r\n");
		return -1;
	}
	else{
		xil_printf("\r\nYour data amount is setup as 0x%08x or %d in
decimal.\r\n", temp, temp);
		*count=temp;
	}
	return 0;
}

int addresSselect(Xuint32 *addr){
	u8 buffer[8];
	int i=0, ret;
	Xuint32 temp=0x0;

	xil_printf("\r\nChoose the starting address for this op\r\n");
	xil_printf("0x0 to 0x00FFFFFF\r\n");
	xil_printf("Terminate your option filling in the 8 hex
chars.\r\n");
	for (i=0;i<8;i++){
		buffer[i] = XUartNs550_RecvByte(UARTBASEADDRESS);
		xil_printf("%c",buffer[i]);
	}
	i=0;
	while ((ret!=0xFF)&(i<8)){
		ret=simpleHex2Dec(buffer[i]);
		temp = (temp<<(4)) + ret;
		i+=1;
	}
	if (ret==0xFF){
		xil_printf("Bad choice of address, try again!\r\n");
		return -1;
	}
	else{
		xil_printf("\r\nYour address is setup as 0x%08x or %d in
decimal.\r\n", temp, temp);
		*addr=temp;
	}
	return 0;
}

int byteRead(Xuint16 *bytetoread){
	u8 buffer[4];
	int i=0, ret;
	Xuint16 temp=0x0;

	for (i=0;i<4;i++){
```

```
            buffer[i] = XUartNs550_RecvByte(UARTBASEADDRESS);
            xil_printf("%c",buffer[i]);
      }
      i=0;
      while ((ret!=0xFF)&(i<4)){
            ret=simpleHex2Dec(buffer[i]);
            temp = (temp<<(4)) + ret;
            i+=1;
      }
      if (ret==0xFF){
            xil_printf("Bad choice of address, try again!\r\n");
            return -1;
      }
      else{
            xil_printf("\r\nYour address is setup as 0x%08x or %d in
decimal.\r\n", temp, temp);
            *bytetoread=temp;
      }
      return 0;
}

void intro(void) {

      xil_printf("\r\nJ3 Flash Operations\r\n");
      xil_printf("This project allows the execution of simple
operations\r\n");
      xil_printf("upon the Nexys 2 FLASH memory.\r\n");

}
```

ANEXO K

Operações executadas na fase de atendimento à interrupção:

Caso Block RAM (endereços de 8 bits)

```
void usb_epp_interrupt_handler(void * baseaddr_p) {

Xuint32 Usb_epp_data;
Xuint32 Usb_epp_address;
Xuint32 Usb_epp_status;
Xuint32 Usb_temp_data;


      Usb_epp_status          =          XIo_In32(USBEPPBASEADDR          +
USB_EPP_STATUS_REG_OFFSET);
      //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",   Usb_epp_status);

      if (Usb_epp_status & USB_EPP_STATUS_READ_REQUESTED)
      {
            //read first the address from where read is requested
            Usb_epp_address     =     XIo_In32     (USBEPPBASEADDR     +
USB_EPP_ADDRESS_REG_OFFSET);

            //read a byte from malloced values, dependent on the counter
value
            Usb_temp_data = data_store[Usb_epp_address];
            //xil_printf("Current  value  of  UsbTempData  for  a  read  is
0x%08X\r\n", Usb_temp_data);

            //filter relevant byte
            switch (data32_counter) {
            case 0:
                  Usb_epp_data = (Usb_temp_data&FILTER0)>>24;
                  data32_counter++;
                  break;
            case 1:
                  Usb_epp_data = (Usb_temp_data&FILTER1)>>16;
                  data32_counter++;
                  break;
            case 2:
                  Usb_epp_data = (Usb_temp_data&FILTER2)>>8;
                  data32_counter++;
                  break;
            case 3:
                  Usb_epp_data = Usb_temp_data&FILTER3;
                  data32_counter=0;
                  break;
            }

            //acknowledge  the  interrupt  by  writing  into  the  specified
address
            XIo_Out32(USBEPPBASEADDR      +      USB_EPP_DATA_REG_OFFSET,
Usb_epp_data);
```

```
            if (UARTONOFF == 1) xil_printf("Data Read Performed, target
address was %d and data read back was 0x%08X\r\n", Usb_epp_address,
Usb_epp_data);
      }

      else if (Usb_epp_status & USB_EPP_STATUS_WRITE_PERFORMED)
      {
            //read first the address from where write was performed
            Usb_epp_address     =     XIo_In32     (USBEPPBASEADDR     +
USB_EPP_ADDRESS_REG_OFFSET);
            //acknowledge the interrupt by reading from the current
address
            Usb_epp_data        =        XIo_In32(USBEPPBASEADDR        +
USB_EPP_DATA_REG_OFFSET);

            //shift and filter relevant data byte, malloc when entire
word has been received
            switch (data32_counter) {
            case 0:
                  Usb_temp_data = (Usb_epp_data<<24)&FILTER0;
                  data32_counter++;
                  break;
            case 1:
                  Usb_temp_data += ((Usb_epp_data<<16)&FILTER1);
                  data32_counter++;
                  break;
            case 2:
                  Usb_temp_data += ((Usb_epp_data<<8)&FILTER2);
                  data32_counter++;
                  break;
            case 3:
                  Usb_temp_data += (Usb_epp_data&FILTER3);
                  data_store[Usb_epp_address] = Usb_temp_data;
                  data32_counter=0;
                  break;
            }

            if (UARTONOFF == 1) xil_printf("Data Write Performed, target
address was %d and data written was 0x%08X\r\n", Usb_epp_address,
Usb_epp_data);

      }
      //re-read the status of the USB-EPP interface
      Usb_epp_status         =         XIo_In32(USBEPPBASEADDR         +
USB_EPP_STATUS_REG_OFFSET);
    //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",     Usb_epp_status);
}
```

ANEXO L

Operações executadas na fase de atendimento à interrupção:

Caso SDRAM Nexys 2 (endereços de 24 bits)

```
void usb_epp_interrupt_handler(void * baseaddr_p) {

Xuint32 Usb_epp_data;
Xuint32 Usb_epp_address;
Xuint32 Usb_epp_status;
Xuint32 Usb_temp_data;
Xuint32 Usb_temp_address;


      Usb_epp_status    =    XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR    +
USB_EPP_STATUS_REG_OFFSET);
      //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",   Usb_epp_status);

      if (Usb_epp_status & USB_EPP_STATUS_READ_REQUESTED)
      {
            //read first the address from where read is requested
            Usb_epp_address = XIo_In32 (XPAR_DIGILENT_USB_EPP_BASEADDR +
USB_EPP_ADDRESS_REG_OFFSET);

            //filter relevant bytes of both data and address
            switch (counter) {
            case 0:
                  Usb_temp_address = (Usb_epp_address<<16)&FILTER1;
      //obtain Upper Byte of address
                  counter++;
                  break;
            case 1:
                  Usb_temp_address += ((Usb_epp_address<<8)&FILTER2);
      //obtain Middle Byte of address
                  counter++;
                  break;
            case 2:
                  Usb_temp_address += (Usb_epp_address&FILTER3);
      //obtain Base Byte of address, 3 Byte address completed
                  counter++;
                  Usb_temp_data      =      XIo_In32(SDRAMBASEADDR      +
Usb_temp_address);     //pull 4 Byte Data Word from SDRAM using 3 Byte
address word
                  if (UARTONOFF == 1) xil_printf("\r\nREAD DATA ADDRESS
0x%08X | WORD 0x%08X\r\n\r\n", Usb_temp_address, Usb_temp_data);
                  Usb_epp_data = (Usb_temp_data&FILTER0)>>24;
            //send Upper Byte of data word
                  break;
            case 3:
                  Usb_epp_data = (Usb_temp_data&FILTER1)>>16;
            //send Middle Byte 1 of data word
                  counter++;
                  break;
            case 4:
                  Usb_epp_data = (Usb_temp_data&FILTER2)>>8;
            //send Middle Byte 2 of data word
```

```
                counter++;
                break;
          case 5:
                Usb_epp_data = Usb_temp_data&FILTER3;
          //send Base Byte of data word, data word completed
                counter=0;
                break;
          }

          //acknowledge the interrupt by writing into the specified
address
          XIo_Out32(XPAR_DIGILENT_USB_EPP_BASEADDR                    +
USB_EPP_DATA_REG_OFFSET, Usb_epp_data);

          if (UARTONOFF == 1) xil_printf("Data Read Performed, target
address was 0x%08X and data read back was 0x%08X\r\n", Usb_epp_address,
Usb_epp_data);
     }

     else if (Usb_epp_status & USB_EPP_STATUS_WRITE_PERFORMED)
     {
          //read first the address from where write was performed
          Usb_epp_address = XIo_In32 (XPAR_DIGILENT_USB_EPP_BASEADDR +
USB_EPP_ADDRESS_REG_OFFSET);
          //acknowledge the interrupt by reading from the current
address
          Usb_epp_data   =   XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR   +
USB_EPP_DATA_REG_OFFSET);

          //shift and filter relevant data byte of both data and
address, malloc when entire word has been received
          switch (counter) {
          case 0:
                Usb_temp_data = (Usb_epp_data<<24)&FILTER0;
                //obtain upper byte of data word
                Usb_temp_address = (Usb_epp_address<<16)&FILTER1;
          //obtain upper byte of address word
                counter++;
                break;
          case 1:
                Usb_temp_data += (Usb_epp_data<<16)&FILTER1;
          //obtain middle byte 1 of data word
                Usb_temp_address += (Usb_epp_address<<8)&FILTER2;
          //obtain middle byte of address word
                counter++;
                break;
          case 2:
                Usb_temp_data += (Usb_epp_data<<8)&FILTER2;
                //obtain middle byte 2 of data word
                Usb_temp_address += Usb_epp_address&FILTER3;
          //obtain base byte of address word, address word complete
                counter++;
                break;
          case 3:
                Usb_temp_data += Usb_epp_data&FILTER3;
                //obtain base byte of data word, data word complete
                XIo_Out32(SDRAMBASEADDR       +       Usb_temp_address,
Usb_temp_data);//push to SDRAM the complete data word, using the complete
address word
```

```
                counter=0;
                  if  (UARTONOFF  ==  1)  xil_printf("\r\nWRITE  DATA
ADDRESS 0x%08X | WORD 0x%08X\r\n\r\n", Usb_temp_address, Usb_temp_data);
                break;
           }

           if (UARTONOFF == 1) xil_printf("Data Write Performed, target
address was 0x%08X and data written was 0x%08X\r\n", Usb_epp_address,
Usb_epp_data);

      }
      //re-read the status of the USB-EPP interface
      Usb_epp_status    =    XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR    +
USB_EPP_STATUS_REG_OFFSET);
    //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",     Usb_epp_status);
}
```

ANEXO M

Operações executadas na fase de atendimento à interrupção:

Caso Flash Nexys 2 (endereços de 24 bits)

```
void usb_epp_interrupt_handler(void * baseaddr_p) {

Xuint32 Usb_epp_data;
Xuint32 Usb_epp_address;
Xuint32 Usb_epp_status;
Xuint16 Usb_temp_word1;
Xuint16 Usb_temp_word2;
Xuint32 Usb_temp_address;


      Usb_epp_status    =    XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR    +
USB_EPP_STATUS_REG_OFFSET);
      //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",   Usb_epp_status);

      if (Usb_epp_status & USB_EPP_STATUS_READ_REQUESTED)
      {
            //read first the address from where read is requested
            Usb_epp_address = XIo_In32 (XPAR_DIGILENT_USB_EPP_BASEADDR +
USB_EPP_ADDRESS_REG_OFFSET);

            //filter relevant bytes of both data and address
            switch (counter) {
            case 0:
                  Usb_temp_address = (Usb_epp_address<<16)&FILTER1;
      //obtain Upper Byte of address
                  counter++;
                  break;
            case 1:
                  Usb_temp_address += ((Usb_epp_address<<8)&FILTER2);
      //obtain Middle Byte of address
                  counter++;
                  break;
            case 2:
                  Usb_temp_address += (Usb_epp_address&FILTER3);
      //obtain Base Byte of address, 3 Byte address completed
                  counter++;
                  read_from_flash(Usb_temp_address,     &Usb_temp_word1);
      //pull 2 Byte Data Word from Flash using 3 Byte address word
                  Usb_epp_data = (Usb_temp_word1&FILTER2)>>8;
            //send Upper Byte of data word
                  break;
            case 3:
                  Usb_epp_data = Usb_temp_word1&FILTER3;
      //send Middle Byte 1 of data word
                  counter++;
                  break;
            case 4:
                  read_from_flash(Usb_temp_address+2, &Usb_temp_word2);
                  Usb_epp_data = (Usb_temp_word2&FILTER2)>>8;
            //send Middle Byte 2 of data word
                  counter++;
```

```
                  break;
            case 5:
                  Usb_epp_data = Usb_temp_word2&FILTER3;
            //send Base Byte of data word, data word completed
                  counter=0;
                  break;
            }

            //acknowledge the interrupt by writing into the specified
address
            XIo_Out32(XPAR_DIGILENT_USB_EPP_BASEADDR                    +
USB_EPP_DATA_REG_OFFSET, Usb_epp_data);

            if (UARTONOFF == 1) xil_printf("Data Read Performed, target
address was 0x%08X and data read back was 0x%08X\r\n", Usb_epp_address,
Usb_epp_data);
      }

      else if (Usb_epp_status & USB_EPP_STATUS_WRITE_PERFORMED)
      {
            //read first the address from where write was performed
            Usb_epp_address = XIo_In32 (XPAR_DIGILENT_USB_EPP_BASEADDR +
USB_EPP_ADDRESS_REG_OFFSET);
            //acknowledge the interrupt by reading from the current
address
            Usb_epp_data   =   XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR   +
USB_EPP_DATA_REG_OFFSET);

            //shift and filter relevant data byte of both data and
address, malloc when entire word has been received
            switch (counter) {
            case 0:
                  Usb_temp_word1 = (Usb_epp_data<<8)&FILTER2;
                  //obtain upper byte of data word 1
                  Usb_temp_address = (Usb_epp_address<<16)&FILTER1;
            //obtain upper byte of address word
                  counter++;
                  break;
            case 1:
                  Usb_temp_word1 += Usb_epp_data&FILTER3;
                  //obtain base byte of data word 1, data word 1 complete
                  Usb_temp_address += (Usb_epp_address<<8)&FILTER2;
            //obtain middle byte of address word
                  counter++;
                  break;
            case 2:
                  Usb_temp_word2 = (Usb_epp_data<<8)&FILTER2;
                  //obtain upper byte of data word 2
                  Usb_temp_address += Usb_epp_address&FILTER3;
            //obtain base byte of address word, address word complete
                  write_to_flash(Usb_temp_address, Usb_temp_word1);
            //since address is complete, send data1 to flash in this
cycle
                  counter++;
                  break;
            case 3:
                  Usb_temp_word2 += Usb_epp_data&FILTER3;
                  //obtain base byte of data word 2, data word 2 complete
```

```
                write_to_flash(Usb_temp_address+2, Usb_temp_word2);
            //send data2 to flash to an address Usb_temp_address+2
                counter=0;
                break;
            }

            if (UARTONOFF == 1) xil_printf("Data Write Performed, target
address was 0x%08X and data written was 0x%08X\r\n", Usb_epp_address,
Usb_epp_data);

      }
      //re-read the status of the USB-EPP interface
      Usb_epp_status    =    XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR    +
USB_EPP_STATUS_REG_OFFSET);
    //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",     Usb_epp_status);
}
```

ANEXO N

Implementação exaustiva da biblioteca de funções quad_spi_if_0.c fornecida pela Digilent para operações com a QuadSPI-Flash presente na Atlys

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

#define UARTBASEADDRESS       XPAR_RS232_UART_1_BASEADDR
#define UARTHIGHADDRESS       XPAR_RS232_UART_1_HIGHADDR
#define UARTCLOCK             XPAR_RS232_UART_1_CLOCK_FREQ_HZ
#define FLASHBASEADDRESS      XPAR_DIGILENT_QUADSPI_CNTLR_BASEADDR
#define FLASHHIGHADDRESS      XPAR_DIGILENT_QUADSPI_CNTLR_HIGHADDR

#define FLASHSIZE       0x01000000
#define SUBSECTORSIZE   0x00001000
#define PAGE_SIZE             256
#define PROGPAGE        128
#define UPPER_BOUNDARY  255
#define LOWER_BOUNDARY  0
#define QUAD                  0x02
#define DUAL                  0x01
#define EXTENDED        0x00

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "xparameters.h"
#include "xuartns550_l.h"

/* Declarations */

      unsigned char verbose;
      Xuint32 mode, address;
      u8 manufactid, memtype, memcapacity, edid, *datalist1, *datalist2,
*datalist;
      int currentaddress, bytecount;

/* LOW LEVEL - Xilinx? */
      Xuint32 Write_Enable (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE,
Xuint32 MODE);
      Xuint32  Read_Status_Register  (Xuint32  QuadSPI_Baseaddr,  Xuint32
DIV_RATE, Xuint32 MODE,Xuint32 NR_OF_BYTES, u8 *data);
      Xuint32  Write_Status_Register  (Xuint32  QuadSPI_Baseaddr,  Xuint32
DIV_RATE, Xuint32 MODE, u8 DATA);
      Xuint32   Read_Flag_Status_Register   (Xuint32   QuadSPI_Baseaddr,
Xuint32 DIV_RATE, Xuint32 MODE, Xuint32 NR_OF_BYTES, u8 *data);
```

```
      Xuint32 Write_Enable (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE,
Xuint32 MODE);
      Xuint32   Write_Volatile_Enhanced_Configuration_Register   (Xuint32
QuadSPI_Baseaddr, Xuint32 DIV_RATE, Xuint32 MODE, u8 DATA);
      Xuint32   Clear_Flag_Status_Register   (Xuint32   QuadSPI_Baseaddr,
Xuint32 DIV_RATE, Xuint32 CURRENT_MODE, Xuint32 MODE);
      Xuint32  Read_Identification  (Xuint32  QuadSPI_Baseaddr,  Xuint32
DIV_RATE,  Xuint32  CURRENT_MODE,  u8  *MANUFACT_ID,  u8  *MEM_TYPE,  u8
*MEM_CAPACITY, u8 *EDID);
      Xuint32 Multiple_IO_Read_Identification (Xuint32 QuadSPI_Baseaddr,
Xuint32 DIV_RATE, Xuint32 CURRENT_MODE, Xuint32 MODE, u8 *MANUFACT_ID, u8
*MEM_TYPE, u8 *MEM_CAPACITY);
      Xuint32  Fast_Read  (Xuint32  QuadSPI_Baseaddr,  Xuint32  DIV_RATE,
Xuint32 CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS, Xuint32 NR_OF_BYTES,
Xuint32 NR_OF_DUMMY_CLKS, u8 DATA[256]);
      Xuint32 Page_Program (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE,
Xuint32 CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS, Xuint32 NR_OF_BYTES,
u8 DATA[256]);
      Xuint32   Subsector_Erase   (Xuint32   QuadSPI_Baseaddr,   Xuint32
DIV_RATE, Xuint32 CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS);
      Xuint32 Sector_Erase (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE,
Xuint32 CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS);
      Xuint32  Bulk_Erase  (Xuint32  QuadSPI_Baseaddr,  Xuint32  DIV_RATE,
Xuint32 CURRENT_MODE, Xuint32 MODE);

/* HIGH LEVEL - Digilent? */
      Xuint32 Check_Initial_Mode (Xuint32 QuadSPI_Baseaddr);
      Xuint32 Quad_SPI_Flash_Test (Xuint32 QuadSPI_Baseaddr);
      Xuint32 Blank_Check_Entire_Memory (Xuint32 QuadSPI_Baseaddr);
      Xuint32 Erase_Entire_Memory (Xuint32 QuadSPI_Baseaddr);
      Xuint32   Blank_Check_Sector   (Xuint32   QuadSPI_Baseaddr,   Xuint32
ADDRESS);
      Xuint32 Erase_Sector (Xuint32 QuadSPI_Baseaddr, Xuint32 ADDRESS);
      Xuint32 Manufact_ID (Xuint32 QuadSPI_Baseaddr);
      Xuint32 Clear_Flags (Xuint32 QuadSPI_Baseaddr);

/* Internal */
      void intro(void);
      void printmenu(void);
      Xuint32 modeselect(void);
      u8 operchoice(void);
      Xuint32 verbochoice(void);
      int pagefiller(Xuint32 nrB, u8* randlist, u8 option);
      Xuint32 designstatusreg(u8 *data);
      int simpleHex2Dec(char hexin);
      int addresSselect(Xuint32 *addr);
      int bytenum(Xuint32 *nrB);
      int memalloc(Xuint32 nrbytes,u8* list);


/* MAIN PROJECT */
int main(void){

      u8 oper, data=0, option;
      Xuint32 newmode, nrofbytes, nrofdummy, ret, Cur_Mode;
      int k, i;

      currentaddress=0;
      bytecount=0;
```

```
	address=0;
	nrofdummy = 10;

	//Initialize RS232 - Set baudrate and number of stop bits
	DONE
	XUartNs550_SetBaud(UARTBASEADDRESS, UARTCLOCK, 115200);
	XUartNs550_SetLineControlReg(UARTBASEADDRESS, XUN_LCR_8_DATA_BITS);

	//Introduction to the app          DONE
	intro();

	//Choosing transmission rate option      DONE
	ret=modeselect();
	if (ret<0){
		xil_printf("App will now close!\r\n");
		return -1;
	}
	else mode=ret;

	// Verbose option       DONE
	ret=verbochoice();
	if (ret<0){
		xil_printf("App will now close!\r\n");
		return -1;
	}
	else verbose=ret;

	Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);

	if (Cur_Mode != mode) {
		if (mode == 2) {
			if  (verbose)  xil_printf("\r\nClearing  QuadSPI  Flash
Flags\r\n");
			Clear_Flags (FLASHBASEADDRESS);
			if   (verbose)   xil_printf("\r\n\r\nWriting   Volatile
Enhanced Configuration Register\r\n");
			Write_Volatile_Enhanced_Configuration_Register
(FLASHBASEADDRESS, 2, Cur_Mode, 0x7F); // quad command input
			Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
		}
		else if (mode == 1) {
			if   (verbose)   xil_printf("Clearing   QuadSPI   Flash
Flags\r\n");
			Clear_Flags (FLASHBASEADDRESS);
			if   (verbose)   xil_printf("\r\n\r\nWriting   Volatile
Enhanced Configuration Register\r\n");
			Write_Volatile_Enhanced_Configuration_Register
(FLASHBASEADDRESS, 2, Cur_Mode, 0xBF); // dual command input
			Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
		}
		else {
			if   (verbose)   xil_printf("Clearing   QuadSPI   Flash
Flags\r\n");
			Clear_Flags (FLASHBASEADDRESS);
			if   (verbose)   xil_printf("\r\n\r\nWriting   Volatile
Enhanced Configuration Register\r\n");
			Write_Volatile_Enhanced_Configuration_Register
(FLASHBASEADDRESS, 2, Cur_Mode, 0xFF); // single command input
			Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
```

```
            }
      }

      while(1){

            //Operations menu
            printmenu();

            //Choosing operation
            oper=operchoice();

            //Switch to perform operation
            switch (oper){


            case 'a':
                        //    A     READ STATUS REGISTER          DONE
                  Read_Status_Register  (FLASHBASEADDRESS,  2,  mode,  1,
&data);             //only returns 0
                  xil_printf("\r\nThe recovered Status Register value is
0x%2x\r\n",data);
                  break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'b':
                        //    B     WRITE STATUS REGISTER         DONE
                  ret=designstatusreg(&data);
                  if (ret<0){
                        xil_printf("App will now close!\r\n");
                        goto exitb; //return -1;
                  }
                  else Write_Status_Register (FLASHBASEADDRESS, 2, mode,
data);            //only returns 0
                  xil_printf("\r\nStatus  Register  successfully  written
with word 0x%02X\r\n",data);
exitb:            break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'c':
                        //    C     READ FLAG STATUS REGISTER     DONE
                  Read_Flag_Status_Register ( FLASHBASEADDRESS, 2, mode,
1, &data);  //only returns 0
                  xil_printf("\r\nThe  recovered  Flag  Status  Register
value is 0x%2x\r\n",data);
                  break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'd':
                        //    D     WRITE ENABLE
      DONE
                  Write_Enable (FLASHBASEADDRESS, 2, mode);
                        //only returns 0
                  xil_printf("\r\nFlash Write Enabled\r\n");
```

```
                break;
          /*-------------------------------------------------------------
----------------------------------------------*/


          case 'e':
                      //    E     WRITE        VOLATILE        ENHANCED
CONFIGURANTION REGISTER
                ret=designstatusreg(&data);
                if (ret<0){
                      xil_printf("Volatile    Enhanced    Configuration
Register creation failed\r\n");
                      goto exite; //return -1;
                }
                Write_Volatile_Enhanced_Configuration_Register
(FLASHBASEADDRESS, 4, mode, data);  //only returns 0
                xil_printf("\r\nVolatile     Enhanced     Configuration
Register written with word 0x%02X\r\n", data);
exite:          break;
          /*-------------------------------------------------------------
----------------------------------------------*/


          case 'f':
                      //    F     CLEAR FLAG STATUS REGISTER
     DONE
                Clear_Flag_Status_Register  (FLASHBASEADDRESS,  2,  mode,
mode);          //only returns 0
                xil_printf("\r\nFlag Status Register cleared.\r\n");
                break;
          /*-------------------------------------------------------------
----------------------------------------------*/


          case 'g':
                      //    G     READ IDENTIFICATION          DONE
                if (mode!= 0x0) {
                      xil_printf("This  op  can  only  be  performed  in
Extended Mode\r\n");
                      goto exitg;
                }
                Read_Identification    (FLASHBASEADDRESS,    2,    mode,
&manufactid, &memtype, &memcapacity, &edid);         //only returns 0
                xil_printf("Your  'Manufacturer's  Identification'  word
is 0x%X\r\n",manufactid);
                xil_printf("Your     'Memory     Type'     word     is
0x%X\r\n",memtype);
                xil_printf("Your    'Memory    Capacity'    word    is
0x%X\r\n",memcapacity);
                xil_printf("Your  'Extended  Device  Identification  is
0x%X\r\n",edid);
exitg:          break;
          /*-------------------------------------------------------------
----------------------------------------------*/


          case 'h':
                      //    H     MULTIPLE IO READ IDENTIFICATION
     DONE
```

```
                if (mode == 0x0){
                      xil_printf("This op can only be performed in Quad
or Dual Mode\r\n");
                      goto exith;
                }
                Multiple_IO_Read_Identification  (FLASHBASEADDRESS,  2,
mode, mode, &manufactid, &memtype, &memcapacity);    //only returns 0
                xil_printf("Your  'Manufacturer's  Identification'  word
is %02x\r\n",manufactid);
                xil_printf("Your     'Memory     Type'     word     is
%02x\r\n",memtype);
                xil_printf("Your    'Memory    Capacity'    word    is
%02x\r\n",memcapacity);
exith:          break;
           /*-------------------------------------------------------------
-----------------------------------------------*/


           case 'i':
                      //    I     FAST READ         DONE
                ret=addressSselect(&address);
                if (ret<0){
                      xil_printf("App  will  now  close,  target  address
selection failed\r\n");
                      goto exiti; //return -1;
                }
                ret=bytenum(&nrofbytes);
                if (ret<0){
                      xil_printf("App will now close, selection of the
number of bytes to program failed\r\n");
                      goto exiti; //return -1;
                }
                if (nrofbytes>(FLASHSIZE-address)){
                      nrofbytes = FLASHSIZE - address;
                      xil_printf("Data  amount  requested  would  target
above  the  Flash  size,  maximum  data  amount  for  this  op  will  be
0x%08X.\r\n", nrofbytes);
                }
                ret=memalloc(nrofbytes,datalist);
                if (ret<0){
                      xil_printf("App will now close, memory allocation
for the page programming failed\r\n");
                      goto exiti; //return -1;
                }
                ret=Fast_Read   (FLASHBASEADDRESS,   2,   mode,   mode,
address, nrofbytes, nrofdummy, datalist);//only returns 0
                xil_printf("You  read  the  following  %d  bytes\r\n",
nrofbytes);
                for (i=0;i<nrofbytes;i++){
                      xil_printf("Byte index %d of %d has value %02x or
%d\r\n", i, nrofbytes-1, datalist[i], datalist[i]);
                }
                free(datalist);
                xil_printf("\r\nFast Read operation completed.\r\n");
exiti:          break;
           /*-------------------------------------------------------------
-----------------------------------------------*/
```

```
            case 'j':
                        //    J     PAGE PROGRAM            DONE C

                  xil_printf("\r\nPage size can be filled with up to 256
bytes, automatically generated or input using stdin\r\n");
                  xil_printf("Data will be placed through Page Program in
up to 2 bursts, up to 128 bytes each transfer for compatibility
reasons\r\n");
                  ret=addresSselect(&address);
                  if (ret<0){
                        xil_printf("App will now close, target address
selection failed\r\n");
                        goto exitj; //return -1;
                  }
                  ret=bytenum(&nrofbytes);
                  if (ret<0){
                        xil_printf("App will now close, selection of the
number of bytes to program failed\r\n");
                        goto exitj; //return -1;
                  }
                  if (nrofbytes>(FLASHSIZE-address)){
                        nrofbytes = FLASHSIZE - address;
                        xil_printf("Data amount requested would target
above the Flash size, maximum data amount for this op will be
0x%08X.\r\n", nrofbytes);
                  }
                  xil_printf("\r\nChoose     the     desired     filling
method:\r\n");
                  xil_printf("Option a) Fill the page to program from
stdin/UART\r\n");
                  xil_printf("Option b) Fill the page to program from a
random value generator\r\n");
                  option = XUartNs550_RecvByte(UARTBASEADDRESS);
                  if (option=='A')(option='a');
                  if (option=='B')(option='b');
                  if ((option!='a')&(option!='b')){
                        xil_printf("\r\nThe pressed key does not map to
one of the options, please try again\r\n");
                        goto exitj;
                  }
                  if (nrofbytes < (PROGPAGE+1)) {
                        ret=memalloc(nrofbytes,datalist);
                        if (ret<0){
                              xil_printf("App  will  now  close,  memory
allocation for the page programming failed\r\n");
                              goto exitj; //return -1;
                        }
                        ret=pagefiller(nrofbytes, datalist, option);
                        if (ret<0){
                              xil_printf("App will now close, page values
design failed\r\n");
                              goto exitj; //return -1;
                        }
                        ret  =  Page_Program  (FLASHBASEADDRESS,  2,  mode,
mode, address, nrofbytes, datalist);
                        if (ret!=0x03){
                              xil_printf("\r\nApp    will    now    close,
'Page_Program' failed.\r\n",ret);
```

```
                              xil_printf("\r\nVolatile            Enhanced
Configuration Register returned was 0x%02X.\r\n", ret);
                              goto exitj; //return -1;
                        }
                        free(datalist);
                  }
                  else {
                        ret=memalloc(PROGPAGE,datalist1);
                        if (ret<0){
                              xil_printf("App  will  now  close,  memory
allocation for the page programming failed\r\n");
                              goto exitj; //return -1;
                        }
                        ret=pagefiller(PROGPAGE, datalist1, option);
                        if (ret<0){
                              xil_printf("App will now close, page values
design failed\r\n");
                              goto exitj; //return -1;
                        }
                        ret=memalloc((nrofbytes-PROGPAGE), datalist2);
                        if (ret<0){
                              xil_printf("App  will  now  close,  memory
allocation for the page programming failed\r\n");
                              goto exitj; //return -1;
                        }
                        ret=pagefiller((nrofbytes - PROGPAGE), datalist2,
option);
                        if (ret<0){
                              xil_printf("App will now close, page values
design failed\r\n");
                              goto exitj; //return -1;
                        }
                        ret  =  Page_Program  (FLASHBASEADDRESS,  2,  mode,
mode, address, PROGPAGE, datalist1);
                        if (ret!=0x03){
                              xil_printf("\r\nApp    will    now    close,
'Page_Program' failed.\r\n",ret);
                              xil_printf("\r\nVolatile            Enhanced
Configuration Register returned was 0x%02X.\r\n", ret);
                              goto exitj; //return -1;
                        }
                        xil_printf("\r\nFirst burst complete\r\n");
                        ret  =  Page_Program  (FLASHBASEADDRESS,  2,  mode,
mode, address+PROGPAGE, nrofbytes - PROGPAGE, datalist2);
                        if (ret!=0x03){
                              xil_printf("\r\nApp    will    now    close,
'Page_Program' failed.\r\n",ret);
                              xil_printf("\r\nVolatile            Enhanced
Configuration Register returned was 0x%02X.\r\n", ret);
                              goto exitj; //return -1;
                        }
                        xil_printf("\r\nSecond burst complete\r\n");
                        free(datalist1);
                        free(datalist2);
                  }
                  xil_printf("\r\nOpp       'Page_Program'       completed
successfully.\r\n");
exitj:            break;
```

```
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'k':
                        //    K     SUBSECTOR ERASE         DONE C
                  ret=addresSselect(&address);
                  if                                                    (
(ret<0)|((address>(8*SUBSECTORSIZE))&(address<(FLASHSIZE-
(8*SUBSECTORSIZE))))){
                        xil_printf("App  will  now  close,  target  address
selection failed\r\n");
                        xil_printf("8   Subsectors   (4096KB   each)   are
present on the Top/Bottom of the QuadSPI-Flash.\r\n");
                        goto exitk; //return -1;
                  }
                  xil_printf("\r\nNow erasing subsector.\r\n");
                  ret = Subsector_Erase (FLASHBASEADDRESS, 2, mode, mode,
address);
                  if (ret!=0x03){
                        xil_printf("\r\nApp  will  now  close,  Subsector
Erase failed.\r\n",ret);
                        xil_printf("\r\nVolatile  Enhanced  Configuration
Register returned was 0x%02X.\r\n", ret);
                        goto exitk; //return -1;
                  }
                  xil_printf("Opp Subsector Erase completed successfully,
subsector that includes address 0x%08X erased.\r\n", address);
exitk:            break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'l':
                        //    L     SECTOR ERASE            DONE
                  ret=addresSselect(&address);
                  if (ret<0){
                        xil_printf("App  will  now  close,  target  address
selection failed\r\n");
                        goto exitl; //return -1;
                  }
                  xil_printf("\r\nNow  erasing  subsector  that  includes
address 0x%08X\r\n", address);
                  ret  =  Sector_Erase  (FLASHBASEADDRESS,  2,  mode,  mode,
address);
                  if (ret!=0x03){
                        xil_printf("\r\nApp will now close, Sector Erase
failed.\r\n",ret);
                        xil_printf("\r\nVolatile  Enhanced  Configuration
Register returned was 0x%02X.\r\n", ret);
                        goto exitl; //return -1;
                  }
                  xil_printf("Opp  Sector  Erase  completed  successfully,
sector that includes address 0x%08X erased.\r\n", address);
exitl:            break;
            /*-------------------------------------------------------------
-----------------------------------------------*/
```

```
            case 'm':
                        //    M     BULK ERASE                    DONE
                  xil_printf("\r\nNow erasing your entire QuadSPI Flash
Memory\r\n");
                  ret = Bulk_Erase (FLASHBASEADDRESS, 2, mode, mode);
                  if (ret!=0x03){
                        xil_printf("\r\nApp will now close, Bulk Erase
failed.\r\n",ret);
                        xil_printf("\r\nVolatile Enhanced Configuration
Register returned was 0x%02X.\r\n", ret);
                        goto exitm; //return -1;
                  }
                  xil_printf("Opp Bulk Erase completed successfully, your
QuadSPI Flash Memory is now empty.\r\n", address);
exitm:            break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'n':
                        //    N     CHECK INITIAL MODE            DONE
                  ret = Check_Initial_Mode (FLASHBASEADDRESS);
                  if (ret == 0x00) xil_printf("\r\nMode checked, QuadSPI-
Flash setup to operate in Extended Mode.\r\n");
                  if (ret == 0x01) xil_printf("\r\nMode checked, QuadSPI-
Flash setup to operate in Dual Mode.\r\n", address);
                  if (ret == 0x02) xil_printf("\r\nMode checked, QuadSPI-
Flash setup to operate in Quad Mode.\r\n", address);
                  if (ret == 0xFFFFFFFF) xil_printf("\r\nMode checked,
QuadSPI-Flash not available.\r\n", address);
                  break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'o':
                        //    O     QUAD SPI FLASH TEST           DONE
                  xil_printf("\r\nQuadSPI-Flash test started.\r\n");
                  ret = Quad_SPI_Flash_Test (FLASHBASEADDRESS);
                  if (ret == 0x00) xil_printf("\nFlash test passed");
                  if (ret == 0x80) xil_printf("\nErasing memory failed");
                  if (ret == 0x70) xil_printf("\nFlash memory not
found");
                  if (ret == 0x60) xil_printf("\nWrite to Flash memory
failed");
                  if (ret == 0x50) xil_printf("\nRead from Flash memory
failed");
                  if (ret == 0x40) xil_printf("\nError on line DQ3");
                  if (ret == 0x30) xil_printf("\nError on line DQ2");
                  if (ret == 0x20) xil_printf("\nError on line DQ1");
                  if (ret == 0x10) xil_printf("\nError on line DQ0");
                  break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'p':
                        //    P     BLANK CHECK ENTIRE MEMORY
      DONE
```

```
                xil_printf("\r\nNow checking entire memory for non-
blank bytes.\r\n");
                ret = Blank_Check_Entire_Memory (FLASHBASEADDRESS);
                if (ret==0x0) xil_printf("\r\nYour QuadSPI-Flash is
empty\r\n");
                else xil_printf("\r\nYour QuadSPI-Flash has data: %d
bytes\r\n", ret);
                break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 'q':
                //    Q     ERASE ENTIRE MEMORY          DONE
                xil_printf("\r\nNow erasing the entire QuadSPI-Flash
memory.\r\n");
                ret = Erase_Entire_Memory (FLASHBASEADDRESS);
                if (ret == 0x0) xil_printf("\r\nQuadSPI-Flash memory
successfully erased.\r\n");
                else xil_printf("\r\nQuadSPI-Flash memory has failed
erasing, Volatile Enhanced Configuration Register returned was
0x%02X\r\n", ret);
                break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 'r':
                //    R     BLANK CHECK SECTOR           DONE
                ret=addresSselect(&address);
                if (ret<0){
                      xil_printf("App will now close, target address
selection failed\r\n");
                      goto exitr; //return -1;
                }
                xil_printf("\r\nNow the sector that includes address
0x%08X for non-blank bytes.\r\n", address);
                ret=Blank_Check_Sector (FLASHBASEADDRESS, address);
                if (ret!=0x0){
                      xil_printf("\r\nNon Blank Bytes found: %d bytes
in sector containing address 0x%08X.\r\n", ret, address);
                      goto exitr; //return -1;
                }
                else xil_printf("\r\nThe sector that includes address
0x%08X is blank\r\n", address);
exitr:          break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 's':
                //    S     ERASE SECTOR                 DONE
                ret=addresSselect(&address);
                if (ret<0){
                      xil_printf("App will now close, target address
selection failed\r\n");
                      goto exits; //return -1;
                }
```

```
                  xil_printf("\r\nNow   erasing   sector   that   includes
address 0x%08X.\r\n", address);
                  ret = Erase_Sector (FLASHBASEADDRESS, address);
                  if  (ret  ==  0x0)  xil_printf("\r\nThe  sector  that
includes address 0x%08X has been erased.\r\n");
                  else {
                        xil_printf("\r\nThe sector that includes address
0x%08X has failed erasing\r\n",address);
                        xil_printf("\r\nVolatile  Enhanced  Configuration
Register returned was 0x%02X\r\n", ret);
                  }
exits:            break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 't':
                        //    T     MANUFACT ID                   DONE
                  ret=Manufact_ID (FLASHBASEADDRESS);
                  if (ret==0xffffffff){
                        xil_printf("App will now close, QuadSPI Flash not
found\r\n");
                        goto exitt; //return -1;
                  }
                  else xil_printf("\r\nYour Manufacturer ID is %08x\r\n",
ret);
exitt:            break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'u':
                        //    U     CLEAR FLAGS                   DONE
                  Clear_Flags (FLASHBASEADDRESS);
                  //only returns 0;
                  xil_printf("\r\nQuadSPI-Flash    Flags    have    been
cleared\r\n");
                  break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            case 'v':
                        //    V     CHANGE MODE                   DONE
                  Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
                  ret=modeselect();
                  if (ret<0){
                        xil_printf("App  will  now  close,  mode  selection
failed\r\n");
                        goto exitv; //return -1;
                  }
                  if (Cur_Mode != ret) {
                        if (ret == 2) {
                              if  (verbose)  xil_printf("Clearing  QuadSPI
Flash Flags\r\n");
                              Clear_Flags (FLASHBASEADDRESS);
                              if   (verbose)   xil_printf("\r\n\r\nWriting
Volatile Enhanced Configuration Register\r\n");
```

```
      Write_Volatile_Enhanced_Configuration_Register   (FLASHBASEADDRESS,
2, Cur_Mode, 0x7F); // quad command input
                              Cur_Mode        =        Check_Initial_Mode
(FLASHBASEADDRESS);
                              mode = QUAD;
                        }
                        else if (ret == 1) {
                              if  (verbose)  xil_printf("Clearing  QuadSPI
Flash Flags\r\n");
                              Clear_Flags (FLASHBASEADDRESS);
                              if   (verbose)   xil_printf("\r\n\r\nWriting
Volatile Enhanced Configuration Register\r\n");

      Write_Volatile_Enhanced_Configuration_Register   (FLASHBASEADDRESS,
2, Cur_Mode, 0xBF); // dual command input
                              Cur_Mode        =        Check_Initial_Mode
(FLASHBASEADDRESS);
                              mode = DUAL;
                        }
                        else {
                              if  (verbose)  xil_printf("Clearing  QuadSPI
Flash Flags\r\n");
                              Clear_Flags (FLASHBASEADDRESS);
                              if   (verbose)   xil_printf("\r\n\r\nWriting
Volatile Enhanced Configuration Register\r\n");

      Write_Volatile_Enhanced_Configuration_Register   (FLASHBASEADDRESS,
2, Cur_Mode, 0xFF); // single command input
                              Cur_Mode        =        Check_Initial_Mode
(FLASHBASEADDRESS);
                              mode = EXTENDED;
                        }
                  }
                  else {
                        xil_printf("\r\nMode  already  in  use,  nothing  to
change.\r\n");
                        mode = Cur_Mode;
                  }
exitv:            break;


            case 'z':
                  //    T     TESTS!
                  xil_printf("\r\n\r\nNothing  to  see  here,  move  along
now...\r\n\r\n");
                  break;
            /*-------------------------------------------------------------
-----------------------------------------------*/


            default:
                  xil_printf("\n\rThe  pressed  key  does  not  map  to  any
available operation, please try again\r\n");
                  break;
            }

      }
```

```
	return 0;
}

void intro(void){ //DONE
	xil_printf("\r\nThis app will allow full configuration and control
of the QuadSPI 128 Mbit Flash present in the Digilent Atlys\r\n");
	xil_printf("First some configuration is required.\r\n");
}

Xuint32 modeselect(void){	//DONE

	u8 newmode;

	xil_printf("\r\nChoose the transmission mode desired: Quad(q),
Dual(d) or Extended(e)\r\n");
	newmode = XUartNs550_RecvByte(UARTBASEADDRESS);
	if (newmode=='Q')(newmode='q');
	if (newmode=='D')(newmode='d');
	if (newmode=='E')(newmode='e');
	if ((newmode!='q')&(newmode!='d')&(newmode!='e')){
		xil_printf("\r\nThe pressed key does not map to one of the
options, please try again\r\n");
		return -1;
	}
	else{
		switch (newmode){
		case 'q': return QUAD;
			break;
		case 'd': return DUAL;
			break;
		case 'e': return EXTENDED;
			break;
		}
	}
}


void printmenu(void){

	xil_printf("\r\nSPI Flash Ops Menu\r\n");
	xil_printf("Choose the desired operation pressing the key
associated with the function\r\n");
	xil_printf("\r\nLow Level Functions:\r\n");
	xil_printf("a) Read_Status_Register\r\n");
	xil_printf("b) Write_Status_Register\r\n");
	xil_printf("c) Read_Flag_Status_Register\r\n");
	xil_printf("d) Write_Enable\r\n");
	xil_printf("e)
Write_Volatile_Enhanced_Configuration_Register\r\n");
	xil_printf("f) Clear_Flag_Status_Register\r\n");
	xil_printf("g) Read_Identification\r\n");
	xil_printf("h) Multiple_IO_Read_Identification\r\n");
	xil_printf("i) Fast_Read\r\n");
	xil_printf("j) Page_Program\r\n");
	xil_printf("k) Subsector_Erase\r\n");
	xil_printf("l) Sector_Erase\r\n");
	xil_printf("m) Bulk_Erase\r\n");
	xil_printf("\n\rHigh Level Digilent Functions\r\n");
	xil_printf("n) Check_Initial_Mode\r\n");
```

```
	xil_printf("o) Quad_SPI_Flash_Test\r\n");
	xil_printf("p) Blank_Check_Entire_Memory\r\n");
	xil_printf("q) Erase_Entire_Memory\r\n");
	xil_printf("r) Blank_Check_Sector\r\n");
	xil_printf("s) Erase_Sector\r\n");
	xil_printf("t) Manufact_ID\r\n");
	xil_printf("u) Clear_Flags\r\n");
	xil_printf("\n\rOther Functions\r\n");
	xil_printf("v) Change QuadSPI-Flash Mode\r\n");
	xil_printf("z) TESTS!!!\r\n");

}

u8 operchoice(void){    //DONE

	u8 operchoice;

	xil_printf("\r\n  Standing by for your input    \r\n");
	operchoice = XUartNs550_RecvByte(UARTBASEADDRESS);
	xil_printf("\r\nThe Selected Operation was: %c) \r\n",operchoice);
	return operchoice;

}

Xuint32 verbochoice(void){    //DONE

	u8 verbo;

	xil_printf("\r\nDo wish a VERBOSE operation? (y OR n)\r\n");
	xil_printf("      Standing by for your input    \r\n");
	verbo = XUartNs550_RecvByte(UARTBASEADDRESS);
	if ((verbo!='y')&(verbo!='n')){
		xil_printf("\r\nThe  pressed  key  does  not  map  to  either
option, please try again\r\n");
		return -1;
	}
	else{
		if (verbo=='y') return 1;
		else return 0;
	}

}

Xuint32 designstatusreg(u8 *data){  //DONE

	u8 buffer[2];
	int ret1, ret2;

	xil_printf("Please input the data to fill this Register\r\n");
	xil_printf("Data should be entered in hex format, 2 chars, to form
a word 0xXX\r\n");
	xil_printf("Enter both chars now:\r\n");
	buffer[1]=XUartNs550_RecvByte(UARTBASEADDRESS);
	buffer[2]=XUartNs550_RecvByte(UARTBASEADDRESS);
	ret1=simpleHex2Dec(buffer[0]);
	ret2=simpleHex2Dec(buffer[1]);
	if ((ret1<0)|(ret2<0)){
		xil_printf("You've  entered  a  char  that  is  not  an  hex
value!\r\n");
```

```
            return -1;
      }
      else{
            *data=(ret1<<4)+ret2;
            xil_printf("you entered 0x%c%c which is %d in base 10",
buffer[1],buffer[2],*data);
      }

      return 0;
}

int simpleHex2Dec(char hexin){     //DONE
      if (hexin=='0')return 0x0;
      if (hexin=='1')return 0x1;
      if (hexin=='2')return 0x2;
      if (hexin=='3')return 0x3;
      if (hexin=='4')return 0x4;
      if (hexin=='5')return 0x5;
      if (hexin=='6')return 0x6;
      if (hexin=='7')return 0x7;
      if (hexin=='8')return 0x8;
      if (hexin=='9')return 0x9;
      if (hexin=='a')return 0xa;
      if (hexin=='b')return 0xb;
      if (hexin=='c')return 0xc;
      if (hexin=='d')return 0xd;
      if (hexin=='e')return 0xe;
      if (hexin=='f')return 0xf;
      if (hexin=='A')return 0xa;
      if (hexin=='B')return 0xb;
      if (hexin=='C')return 0xc;
      if (hexin=='D')return 0xd;
      if (hexin=='E')return 0xe;
      if (hexin=='F')return 0xf;
      else return -1;
}

int addresSselect(Xuint32 *addr){  //DONE

      u8 buffer[8];
      int i=0, ret;
      Xuint32 temp=0x00000000;

      xil_printf("\r\nChoose the starting address for this op (0x00000000
to 0x01FFFFFF):\r\n");
      xil_printf("Terminate your option filling in the 8 hex
chars.\r\n");

      for (i=0;i<8;i++){
            buffer[i] = XUartNs550_RecvByte(UARTBASEADDRESS);
            xil_printf("%c",buffer[i]);
      }

      i=0;

      while ((ret!=-1)&(i<8)){
            ret=simpleHex2Dec(buffer[i]);
            temp = (temp<<(4)) + ret;
            i+=1;
```

```
	}

	xil_printf("Your address is setup as 0x%08x or %d in decimal.\r\n",
temp, temp);

	if ((temp < 0x00)|(i < 1)){
		xil_printf("Bad choice of address, try again!\r\n");
		return -1;
	}
	else{
		if (temp>0x01FFFFFF){
			xil_printf("\r\nRemember that the top address for this
op must be 0x01FFFFFF\r\n");
			xil_printf("\r\nYour option failed this prerequisite,
app will now exit\r\n");
			return -1;
		}
		else{
			*addr=temp;
		}
	}
	return 0;

}

int pagefiller(Xuint32 nrB, u8* randlist, u8 option){//DONE

	int i;
	u8  received[2];
	int ret1, ret2;

	switch (option){
	case 'a':
		for (i=0;i<nrB;i++){
			xil_printf("Enter a new byte word using hex
representation, index %d of %d:\r\n",i,PAGE_SIZE-1);
			received[0] = XUartNs550_RecvByte(UARTBASEADDRESS);
			xil_printf("%c",received[0]);
			received[1] = XUartNs550_RecvByte(UARTBASEADDRESS);
			xil_printf("%c",received[1]);
			ret1=simpleHex2Dec(received[0]);
			ret2=simpleHex2Dec(received[1]);
				if ((ret1<0)|(ret2<0)){
					xil_printf("You've entered a char that is
not an hex value!\r\n");
					return -1;
				}
				else{
					randlist[i]=(ret1<<4)+ret2;
					xil_printf("You entered 0x%c%c which is %d
in base 10\r\n", received[0],received[1],randlist[i]);
				}
		}
		break;
	case 'b':
		srand((Xuint32)time(NULL));
			//random seed setup
		for (i=0;i<nrB;i++){
			//random filler generation
```

```
                  randlist[i]                                                    =
(u8)((char)(LOWER_BOUNDARY+(int)((double)(UPPER_BOUNDARY-
LOWER_BOUNDARY+1)*rand()/(RAND_MAX+1.0))));
                  xil_printf("Value index %d of %d is: %02x or %d\r\n",
i, nrB-1, randlist[i], randlist[i]);
            }
            break;
      }

      return 0;
}

int bytenum(Xuint32 *nrB){
      u8 buffer[2];
      int ret1, ret2;

      xil_printf("Enter the amount of byte words you wish to
read/write\r\n");
      xil_printf("Data should be entered in hex format (hence the desired
amount minus 1)\r\n");
      xil_printf("2 chars, to form a word 0xXX (0x00 upto 0xFF)\r\n");
      xil_printf("Enter both chars now:\r\n");
      buffer[0]=XUartNs550_RecvByte(UARTBASEADDRESS);
      xil_printf("%c",buffer[0]);
      buffer[1]=XUartNs550_RecvByte(UARTBASEADDRESS);
      xil_printf("%c",buffer[1]);
      ret1=simpleHex2Dec(buffer[0]);
      ret2=simpleHex2Dec(buffer[1]);
      if ((ret1<0)|(ret2<0)){
            xil_printf("You've entered a char that is not an hex
value!\r\n");
            return -1;
      }
      else{
            *nrB=((ret1<<4)+ret2)+1;
            xil_printf("\r\nYou entered 0x%C%C thus meaning that the
number of bytes to read/write is %d in base 10",
buffer[0],buffer[1],*nrB);
      }

      return 0;
}


int memalloc(Xuint32 nrbytes,u8* list){

      list = (u8*)malloc(sizeof(u8)*nrbytes);
      //array memory allocation
      if (list == NULL){
            xil_printf("\r\nFailed allocating memory!\r\n");
            return -1;
      }
      return 0;
}
```

ANEXO O

Implementação exaustiva de funções de I/O e teste da DDR2-SDRAM presente na Atlys

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

#define DDRBASEADDRESS  XPAR_MCB_DDR2_MPMC_BASEADDR
#define DDRHIGHADDRESS  XPAR_MCB_DDR2_MPMC_HIGHADDR
#define     UARTBASEADDRESS   XPAR_RS232_UART_1_BASEADDR
#define UARTHIGHADDRESS XPAR_RS232_UART_1_HIGHADDR
#define UARTCLOCK       XPAR_RS232_UART_1_CLOCK_FREQ_HZ
#define DATAWIDTH32     32
#define DATAWIDTH8            8
#define TESTWORD32            0xF0A53CC3
#define TESTWORD8       0xA3
#define NULL32                0x00000000
#define NULL8                 0x00
#define DDRSIZE               0x08000000

#include <stdio.h>
#include "xutil.h"
#include <stdlib.h>
#include <time.h>
#include "string.h"
#include "xenv_standalone.h"
#include "xintc.h"
#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"
#include "xuartns550_l.h"

unsigned char verbose;

Xuint32   memory_test   (unsigned   int   MEM_BASEADDR,   unsigned   int
MEM_HIGHADDR, unsigned int DATA_WIDTH);

void intro(void);
Xuint32 verbo(void);
void options(void);
int simpleHex2Dec(char hexin);
Xuint32 testmemory32(Xuint32 testword, Xuint32 testaddress);
int read32word(Xuint32 *word32);

int main(void){
```

```
	u8 opt, wordorigin;
	Xuint32 newtest32, address, datacount, ret, word8, word32;
	int err=0, i;


	//Startup and setup UART
	XUartNs550_SetBaud(UARTBASEADDRESS, UARTCLOCK, 115200);
	XUartNs550_SetLineControlReg(UARTBASEADDRESS, XUN_LCR_8_DATA_BITS);

	//Introduce program
	intro();
	//Verbose?
	verbose = verbo();
	if (verbose<0) {
		xil_printf("\r\nYou have pressed a key that does not map to
an available option, please try again!\r\n");
		return -1;
	}

	//Erase all DDR2 Memory
	xil_printf("Now erasing all DDR2 Memory, to start with a clean
memory.\r\n");
	for (i=0;i<DDRSIZE;i=i+4) {
		XIo_Out32(DDRBASEADDRESS+i, NULL32);
	}

	//core
	while (1){
		//Present program options
		options();
		//Read option
		opt = XUartNs550_RecvByte(UARTBASEADDRESS);
		//Execute Select Option

		xil_printf("You chose operation %c\r\n", opt);

		switch (opt){

		case 'a':   //Digilent Memory Test
			DONE
			memory_test(DDRBASEADDRESS, DDRHIGHADDRESS, 32);
			break;
		/*-------------------------------------------------------------
-----------------------------------------------*/


		case 'b':   //Write/Read Test Entire Memory
			DONE
			xil_printf("\r\n  You  may  at  this  point  opt  to
write/read back the standard word or your own word\r\n");
			xil_printf("The standard test word is 0xF0A53CC3 or you
may enter your own (a or b)\r\n");
			wordorigin=XUartNs550_RecvByte(UARTBASEADDRESS);
			if (wordorigin=='A')(wordorigin='a');
			if (wordorigin=='B')(wordorigin='b');
			if ((wordorigin!='a')&(wordorigin!='b')){
				xil_printf("You have pressed a key that does not
map to an available option, please try again!\r\n");
				goto exitb;
```

```
                }
                else {
                      err=0;
                      if (wordorigin=='a'){
                            for (i=0;i<DDRSIZE;i+=4){

      ret=testmemory32(DDRBASEADDRESS+i,TESTWORD32);
                                  if (ret!=0){
                                        err+=1;
                                        xil_printf("Error  detected  at
address  0x%08X,  value  read  was  0x%08X  when  0x%08X  was  expected\r\n",
address, ret, TESTWORD32);
                                  }
                            }
                            if (err!=0){
                                  xil_printf("Memory  Test  Failed,  %d
errors identified\r\n", err);
                                  break;
                            }
                            else     xil_printf("Memory     Test     was
successful\r\n");
                      }
                      else{
                            xil_printf("\r\nChoose   the   test   word
desired for this op (0x00000000 to 0xFFFFFFFF)\r\n");
                            xil_printf("Do   fill   in   the   8   hex
chars\r\n");
                            ret=read32word(&newtest32);
                            if (ret!=0){
                                  xil_printf("Reading   of   new   word
failed, try again\r\n", ret);
                                  goto exitb;
                            }
                            for (i=0;i<DDRSIZE;i+=4){
                                  ret=testmemory32(DDRBASEADDRESS+i,
newtest32);
                                  if (ret!=0){
                                        err+=1;
                                        xil_printf("Error  detected  at
address  0x%08X,  value  read  was  0x%08X  when  0x%08X  was  expected\r\n",
address, ret, newtest32);
                                  }
                            }
                            if (err!=0){
                                  xil_printf("Memory  Test  Failed,  %d
errors identified\r\n", err);
                                  break;
                            }
                            else     xil_printf("Memory     Test     was
successful\r\n");
                      }
                }
exitb:          break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 'c':   //Erase     Data
          DONE
```

```
                xil_printf("\r\nChoose the address for this op
(0x00000000 up to 0x07FFFFFF)\r\n");
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&address);
                if ((ret!=0)|(address<0)|(address>0x07FFFFFF)){
                      xil_printf("Address selection failed, try
again\r\n", ret);
                      goto exitc;
                }
                xil_printf("\r\nChoose the data amount for this op
(0x00000000 up to 0x%08X)\r\n", DDRSIZE-address);
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&datacount);
                if ((ret!=0)|(datacount<0)){
                      xil_printf("Reading of data amount failed, try
again\r\n", ret);
                      goto exitc;
                }
                if (datacount>(DDRSIZE-address)){
                      datacount = DDRSIZE - address;
                      xil_printf("Data amount requested would target
above the DDR size, maximum data amount for this op will be 0x%08X.\r\n",
datacount);
                }
                for(i=0;i<datacount;i=i+4) {
                      XIo_Out32(DDRBASEADDRESS+address+i, 0x00000000);
                }
                xil_printf("Data successfully erased: starting at
address 0x%08X, 0x%08X bytes\r\n", address, datacount);
exitc:          break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 'd':   //Write Data
     DONE
                xil_printf("\r\nChoose the address for this op
(0x00000000 up to 0x07FFFFFF)\r\n");
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&address);
                if ((ret!=0)|(address<0)|(address>0x07FFFFFF)){
                            xil_printf("Address selection failed, try
again\r\n", ret);
                            goto exitd;
                }
                xil_printf("\r\nChoose the data amount for this op
(0x00000000 up to 0x%08X) bytes\r\n", DDRSIZE-address);
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&datacount);
                if ((ret!=0)|(datacount<0)){
                      xil_printf("Reading of data amount failed, try
again\r\n", ret);
                      goto exitd;
                }
                if (datacount>(DDRSIZE-address)){
                      datacount = DDRSIZE - address;
                      xil_printf("Data amount requested would target
above the DDR size, maximum data amount for this op will be 0x%08X.\r\n",
datacount);
```

```
                }
                xil_printf("\r\nInsert every 32 bit word now, up to the
data amount bytes chosen\r\n");
                for(i=0;i<datacount;i=i+4) {
                      ret=read32word(&word32);
                      if ((ret!=0)|(word32<0)){
                            xil_printf("Reading  of  word  failed,  try
again\r\n", ret);
                            goto exitd;
                      }
                      else XIo_Out32(DDRBASEADDRESS+address+i, word32);
                }
                xil_printf("Data  successfully  written:  starting  at
address 0x%08X, 0x%08X bytes\r\n", address, datacount);
exitd:          break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 'e':   //Read Data                                  DONE
                xil_printf("\r\nChoose   the   address   for   this   op
(0x00000000 up to 0x07FFFFFF)\r\n");
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&address);
                if ((ret!=0)|(address<0)|(address>0x07FFFFFF)){
                      xil_printf("Address    selection    failed,    try
again\r\n", ret);
                      goto exite;
                }
                xil_printf("\r\nChoose  the  data  amount  for  this  op
(0x00000000 up to 0x%08X) bytes\r\n", DDRSIZE-address);
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&datacount);
                if ((ret!=0)|(datacount<0)){
                      xil_printf("Reading  of  data  amount  failed,  try
again\r\n", ret);
                      goto exite;
                }
                if (datacount>(DDRSIZE-address)){
                      datacount = DDRSIZE - address;
                      xil_printf("Data  amount  requested  would  target
above the DDR size, maximum data amount for this op will be 0x%08X.\r\n",
datacount);
                }
                for(i=0;i<datacount;i=i+4) {
                      word32 = XIo_In32(DDRBASEADDRESS+address+i);
                      xil_printf("Address    0x%08X,    32    bit    word
0x%08X\r\n", address+i, word32);
                }
                xil_printf("Data successfully read: starting at address
0x%08X, 0x%08X bytes\r\n", address, datacount);
exite:          break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 'f':   //Generate And Write Random Words
                rand();                                    //workaround
to avoid sequence repetition
```

```
                srand(2);
                xil_printf("\r\nChoose the address for this op
(0x00000000 up to 0x07FFFFFF)\r\n");
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&address);
                if ((ret!=0)|(address<0)|(address>0x07FFFFFF)){
                      xil_printf("Address selection failed, try
again\r\n", ret);
                      goto exitf;
                }
                xil_printf("\r\nChoose the data amount for this op
(0x00000000 up to 0x%08X) bytes\r\n", DDRSIZE-address);
                xil_printf("Do fill in the 8 hex chars\r\n");
                ret=read32word(&datacount);
                if ((ret!=0)|(datacount<0)){
                      xil_printf("Reading of data amount failed, try
again\r\n", ret);
                      goto exitf;
                }
                if (datacount>(DDRSIZE-address)){
                      datacount = DDRSIZE - address;
                      xil_printf("Data amount requested would target
above the DDR size, maximum data amount for this op will be 0x%08X.\r\n",
datacount);
                }
                xil_printf("\r\nNow writing your data\r\n");
                for(i=0;i<datacount;i=i+4) {
                      word32=rand();
                      XIo_Out32(DDRBASEADDRESS+address+i, word32); //
with RAND_MAX 0x7FFFFFFF
                      xil_printf("Address 0x%08X, 32 bit word
0x%08X\r\n", address+i, word32);
                }
                xil_printf("Data successfully written: starting at
address 0x%08X, 0x%08X bytes\r\n", address, datacount);
exitf:          break;
          /*-------------------------------------------------------------
-----------------------------------------------*/


          case 'g':   //Blank Check Entire Memory
                      DONE
                xil_printf("Now checking the entire DDR Memory for non-
null bytes.\r\n");
                err=0;
                for (i=0;i<DDRSIZE;i++) {
                      word8 = XIo_In8(DDRBASEADDRESS+i);
                      if (word8!=NULL8) {
                            err+=1;
                            xil_printf("Non-null value detected at
address 0x%08X, value read was 0x%02X when 0x%02X was expected\r\n", i,
ret, NULL8);
                      }
                }
                if (err!=0){
                      xil_printf("Memory is not empty, %d data bytes
identified\r\n", err);
                }
                else xil_printf("Memory is empty\r\n");
```

```
                break;
          /*-------------------------------------------------------------
----------------------------------------------*/


          case 'h':         //Erase Entire Memory
                xil_printf("\r\nStandby,     erasing     entire     DDR2
Memory...\r\n");
                for (i=0;i<DDRSIZE;i=i+4) {
                      XIo_Out32(DDRBASEADDRESS+i, NULL32);
                }
                xil_printf("DDR2 Memory successfully erased.\r\n\r\n");
                break;
          /*-------------------------------------------------------------
----------------------------------------------*/


          case 'z':                     //TESTS!!!!
                rand();
                srand(2);
                xil_printf("0x%08X", rand());
                break;


          default:
                xil_printf("\r\nYou have pressed a key that does not
map to an available option, please try again!\r\n");
                break;
          }
     }
     return 0;
}

void intro (void) {
     xil_printf("\r\nThis   program   is   designed   to   control   and
manipulate\r\n");
     xil_printf("the  onboard  DDR2  RAM  memory  inside  the  Digilent
Atlys\r\n");
}

Xuint32 verbo(void) {

     u8 opt;

     xil_printf("\r\nDo you wish a verbose operation (y or n)?\r\n");
     opt=XUartNs550_RecvByte(UARTBASEADDRESS);
     if (opt=='Y')(opt='y');
     if (opt=='N')(opt='n');
     if ((opt!='y')&(opt!='n'))return -1;
     else {
          if (opt=='y')return 1;
          else return 0;
     }
}

void options (void) {
     xil_printf("\r\nThe following are the RAM operations available for
selection\r\n");
```

```
	xil_printf("a)  Digilent  Memory  Test  (data  is  changed  in
Memory)\r\n");
	xil_printf("b) Write/Read Test Entire Memory\r\n");  //
	xil_printf("c) Erase Data\r\n");
	xil_printf("d) Write Data\r\n");
	xil_printf("e) Read Data\r\n");
	xil_printf("f) Generate And Write Random Words\r\n");
	xil_printf("g) Blank Check Entire Memory\r\n");
	xil_printf("h) Erase Entire Memory\r\n");
	xil_printf("\r\nEnter the character that matches your option\r\n");
}

int simpleHex2Dec(char hexin){     //DONE
	if (hexin=='0')return 0x0;
	if (hexin=='1')return 0x1;
	if (hexin=='2')return 0x2;
	if (hexin=='3')return 0x3;
	if (hexin=='4')return 0x4;
	if (hexin=='5')return 0x5;
	if (hexin=='6')return 0x6;
	if (hexin=='7')return 0x7;
	if (hexin=='8')return 0x8;
	if (hexin=='9')return 0x9;
	if (hexin=='a')return 0xa;
	if (hexin=='b')return 0xb;
	if (hexin=='c')return 0xc;
	if (hexin=='d')return 0xd;
	if (hexin=='e')return 0xe;
	if (hexin=='f')return 0xf;
	if (hexin=='A')return 0xa;
	if (hexin=='B')return 0xb;
	if (hexin=='C')return 0xc;
	if (hexin=='D')return 0xd;
	if (hexin=='E')return 0xe;
	if (hexin=='F')return 0xf;
	else return -1;
}

Xuint32 testmemory32(Xuint32 testaddress, Xuint32 testword){

	Xuint32 readdata;

	XIo_Out32(testaddress,testword);
	readdata=XIo_In32(testaddress);
	if (readdata!=testword)return readdata;
	else return 0x00;
}

int read32word(Xuint32 *word32){
	u8 buffer[8];
	int i, ret;
	Xuint32 temp=0x00000000;

	for (i=0;i<8;i++){
		buffer[i] = XUartNs550_RecvByte(UARTBASEADDRESS);
		xil_printf("%c", buffer[i]);
	}
	i=0;
	while ((ret!=-1)&(i<8)){
```

```
            ret=simpleHex2Dec(buffer[i]);
            if (ret<0){
                  xil_printf("One of the chars entered does not map to an
hex char\r\n");
                  return -1;
            }
            temp = (temp<<(4)) + ret;
            i+=1;
      }
      *word32 = temp;
      return 0;
}
```

ANEXO P

Operações executadas na fase de atendimento à interrupção:

Caso QuadSPI-Flash (endereços de 24 bits, escrita pontual)

```
void usb_epp_interrupt_handler(void * baseaddr_p) {
      Xuint32 Usb_epp_data;
      Xuint32 Usb_epp_address;
      Xuint32 Usb_epp_status;
      Xuint32 Usb_temp_data;
      Xuint32 Usb_temp_address;
      Xuint32 Usb_base_address;

      Usb_epp_status        =        XIo_In32(USBEPPBASEADDRESS        +
USB_EPP_STATUS_REG_OFFSET);
      //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",   Usb_epp_status);

      if (Usb_epp_status & USB_EPP_STATUS_READ_REQUESTED)
      {
            //read first the address from where read is requested
            Usb_epp_address    =    XIo_In32    (USBEPPBASEADDRESS    +
USB_EPP_ADDRESS_REG_OFFSET);

            //filter relevant bytes of both data and address
            switch (counter) {
            case 0:
                  Usb_temp_address = (Usb_epp_address<<24)&FILTER0;
      //obtain Upper Byte of address
                  counter++;
                  break;
            case 1:
                  Usb_temp_address += (Usb_epp_address<<16)&FILTER1;
      //obtain Middle Byte 1 of address
                  counter++;
                  break;
            case 2:
                  Usb_temp_address += ((Usb_epp_address<<8)&FILTER2);
      //obtain Middle Byte 2 of address
                  counter++;
                  break;
            case 3:
                  Usb_temp_address += (Usb_epp_address&FILTER3);
      //obtain Base Byte of address, 4 Byte address completed
                  counter++;
                  ret   =   Fast_Read   (FLASHBASEADDRESS,   2,   2,   2,
Usb_temp_address, 4, 10, data_store);    //pull  4  Byte  Data  Word  from
Flash using 4 Byte address word
                  if  (UARTONOFF  ==1  )  xil_printf("Returned  data  was
0x%02X%02X%02X%02X\r\n",data_store[0],    data_store[1],    data_store[2],
data_store[3]);
                  Usb_epp_data = data_store[0];
            //send Upper Byte of data word
                  break;
            case 4:
                  Usb_epp_data = data_store[1];
            //send Middle Byte 1 of data word
```

```
                  counter++;
                  break;
            case 5:
                  Usb_epp_data = data_store[2];
            //send Middle Byte 2 of data word
                  counter++;
                  break;
            case 6:
                  Usb_epp_data = data_store[3];
            //send Base Byte of data word, data word sent
                  counter=0;
                  break;
            }

            //acknowledge the interrupt by writing into the specified
address
            XIo_Out32(USBEPPBASEADDRESS    +    USB_EPP_DATA_REG_OFFSET,
Usb_epp_data);

            if (UARTONOFF ==1 ) xil_printf("Data Read Performed, target
address was 0x%08X and data read back was 0x%08X\r\n", Usb_epp_address,
Usb_epp_data);
      }

      else if (Usb_epp_status & USB_EPP_STATUS_WRITE_PERFORMED)
      {
            //read first the address from where write was performed
            Usb_epp_address    =    XIo_In32    (USBEPPBASEADDRESS    +
USB_EPP_ADDRESS_REG_OFFSET);
            //acknowledge the interrupt by reading from the current
address
            Usb_epp_data       =       XIo_In32(USBEPPBASEADDRESS       +
USB_EPP_DATA_REG_OFFSET);

            //shift and filter relevant data byte of both data and
address, store when entire word has been received
            switch (counter) {
            case 0:
                  Usb_temp_address  = (Usb_epp_address<<24)&FILTER0;
            //obtain upper byte of address word
                  counter++;
                  data_store[progcounter] = Usb_epp_data;
                        //obtain upper byte of data word
                  progcounter++;
                  break;
            case 1:
                  Usb_temp_address  += (Usb_epp_address<<16)&FILTER1;
            //obtain middle byte 1 of address word
                  counter++;
                  data_store[progcounter] = Usb_epp_data;
                        //obtain middle byte 1 of data word
                  progcounter++;
                  break;
            case 2:
                  Usb_temp_address  += (Usb_epp_address<<8)&FILTER2;
            //obtain middle byte 2 of address word
                  counter++;
                  data_store[progcounter] = Usb_epp_data;
                        //obtain middle byte 2 of data word
```

```
                progcounter++;
                break;
          case 3:
                Usb_temp_address  += Usb_epp_address&FILTER3;
                //obtain  base  byte  of  address  word,  address  word
complete
                data_store[progcounter] = Usb_epp_data;
                      //obtain base byte of data word
                progcounter++;
                if (progcounter == 4) {
                      Usb_base_address = Usb_temp_address;
                }
                if ((Usb_temp_address & LASTWORD)==LASTWORD) {
                //check if it's the last word
                      lastwordflag = 1;
                      Usb_temp_address -= LASTWORD;
                }
                if  (UARTONOFF  ==1  )  xil_printf("Lastword  value  is
%d\r\n", lastwordflag);
                if ((progcounter == PROGPAGE)||(lastwordflag == 1)) {
          //if  it's  the  last  word  OR  not  the  last  word  but  malloced
buffer is full
                      ret  =  Page_Program  (FLASHBASEADDRESS,  2,  2,  2,
Usb_base_address , progcounter , data_store);  //push   to   Flash   the
complete half-page data, using the complete address word
                      if (UARTONOFF ==1 ) {
                            if (ret !=0x03) xil_printf("PP failed with
code 0x%02X\r\n", ret);
                            else xil_printf("PP worked\r\n");
                            xil_printf("progcounter   has   value   %d,
address was 0x%08X\r\n", progcounter, Usb_base_address);
                      }
                      progcounter = 0;
                      lastwordflag = 0;
                }
                counter=0;
                break;
          }

          if (UARTONOFF ==1 ) xil_printf("Data Write Performed, target
address  was  0x%08X  and  data  written  was  0x%08X\r\n",  Usb_epp_address,
Usb_epp_data);
     }
     //re-read the status of the USB-EPP interface
     Usb_epp_status        =        XIo_In32(USBEPPBASEADDRESS        +
USB_EPP_STATUS_REG_OFFSET);
   //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",     Usb_epp_status);
}
```

ANEXO Q

Funções de escrita ou leitura de dados da QuadSPI-Flash através de USB-EPP:

operações realizadas no computador de uso geral

```
void Send4BFPGA()
{
      unsigned int memind=0x0, memindoriginal = 0x0, memind_t=0x0;
      unsigned int memval=0x0, memval_t=0x0;

      /* Send 32 bits to Atlys using DoPutReg() function */

      memindoriginal = atoi(szRegister);             //read from stdin
      memind = memindoriginal + LASTWORD;

      if (memindoriginal<FLASHSIZE-4) {
            memval = atoi(szByte);                                  //read
from stdin
            memind_t = (FILTER0&memind)>>24;
            _itoa(memind_t, szRegister, 10);
            memval_t = (FILTER0&memval)>>24;
            _itoa(memval_t, szByte, 10);
            DoPutReg();                                                    //
First Byte sent
            memind_t = (FILTER1&memind)>>16;
            _itoa(memind_t, szRegister, 10);
            memval_t = (FILTER1&memval)>>16;
            _itoa(memval_t, szByte, 10);
            DoPutReg();                                                    //
Second Byte sent
            memind_t = (FILTER2&memind)>>8;
            _itoa(memind_t, szRegister, 10);
            memval_t = (FILTER2&memval)>>8;
            _itoa(memval_t, szByte, 10);
            DoPutReg();                                                    //
Third Byte sent
            memind_t = FILTER3&memind;
            _itoa(memind_t, szRegister, 10);
            memval_t = FILTER3&memval;
            _itoa(memval_t, szByte, 10);
            DoPutReg();                                                    //
Fourth Byte sent
            printf("Complete. QUADSPI Flash index %d set with value %u /
0x%08X.\n", memindoriginal ,memval, memval);
      }
      else {
            printf("Failed. You tried to address outside the limits of
Flash memory:\n");
            printf("address from 0x0 to 0x00FFFFFF.\n");
      }
 }

/*********************************************************************
/
```

```
void Get4BFPGA(){

      unsigned int memind=0x0, memindoriginal = 0x0, memind_t=0x0;
      unsigned int memval=0x0,memval_t=0x0;

      /* Receive 32 bits from Atlys using DoGetReg() function */

      memindoriginal = atoi(szRegister);             //read from stdin
      memind = memindoriginal;

      if (memindoriginal<FLASHSIZE-4) {
            memind_t = (FILTER0&memind)>>24;
            //printf("First octet of address is 0x%02X\n",memind_t);
            _itoa(memind_t, szRegister, 10);
            DoGetReg();                                                  //
First Byte of address word sent
            memind_t = (FILTER1&memind)>>16;
            //printf("First octet of address is 0x%02X\n",memind_t);
            _itoa(memind_t, szRegister, 10);
            DoGetReg();                                                  //
Second Byte of address word sent
            memind_t = (FILTER2&memind)>>8;
            //printf("First octet of address is 0x%02X\n",memind_t);
            _itoa(memind_t, szRegister, 10);
            DoGetReg();                                                  //
Third Byte of address word sent
            memind_t = FILTER3&memind;
            //printf("First octet of address is 0x%02X\n",memind_t);
            _itoa(memind_t, szRegister, 10);
            DoGetReg();                                                  //
Fourth Byte of address word sent
            //printf("first octet is 0x%02X\n",idData);
            memval = (idData<<24)&FILTER0;                       // First
Byte of data word received
            DoGetReg();                                                  //
Second Byte of data word received
            //printf("second octet is 0x%02X\n",idData);
            memval += (idData<<16)&FILTER1;
            DoGetReg();                                                  //
Third Byte of data word received
            //printf("third octet is 0x%02X\n",idData);
            memval += (idData<<8)&FILTER2;
            DoGetReg();                                                  //
Fourth Byte of data word received
            //printf("fourth octet is 0x%02X\n",idData);
            memval += idData&FILTER3;
            printf("Complete. QUADSPI Flash index %d reported value %u /
0x%08X.\n", memindoriginal ,memval, memval);
      }
      else {
            printf("Failed. You tried to address outside the limits of
Flash memory:\n");
            printf("address from 0x0 to 0x00FFFFFF.\n");
      }
}

/*********************************************************************
/
```

```
void SendDataToAtlys() {

	FILE *filein;
	unsigned int regval, index, address, count, i, valuefromfile,
valuefromfileahead, flag;
	int j;
	char temp;
	float complete, timeelapsed;
	LARGE_INTEGER freq, starttick, endtick;

	if((filein = fopen(szFile,"r"))==NULL) {
		printf("Error opening input file!\n");
		goto lFail;
	}

	printf("Opened file was ' %s '\n", szFile);

	address = atoi(szRegister);                                    //read
from stdin, address is the point at which data will start to be written
	count = atoi(szByte);
	//read from stdin, count is the ammount of 32 bit words from the
dataTOatlys.txt file that will be written
	if ((address+(count*4))>=FLASHSIZE){
		printf("Considering both address and data amount you'd be
addressing outside the Flash area\n");
		count = FLASHSIZE-address-1;
		printf("To avoid this data amount will be cropped to %d.\n",
count);
	}
	printf("Sending your values to Atlys Flash starting from address
0x%08X, an amount of %d words will be written from the value list.\n",
address, count);
	printf("Please hold...\n");

	if (fscanf(filein, "%u\n", &valuefromfile)!=EOF) {
                                        //if file has at least one item
	i=0;
	j=0;
	flag=0;

	QueryPerformanceFrequency(&freq);
	QueryPerformanceCounter(&starttick);

		while (i<(WORDBYTES*count)) {  //(!feof(filein))

			if		((fscanf(filein,		"%u\n",
&valuefromfileahead)==EOF)||(i==(WORDBYTES*(count-1)))) {
				//test ahead for EOF
				address = address + LASTWORD;
				flag = 1;
			}



                                    //if there's another next in line, do
not change address
```

```
                //printf("Address will be 0x%08X / %u\n", address+i,
address+i);
                //printf("Value read 0x%08X / %u\n", valuefromfile,
valuefromfile);

                index = (FILTER0&(address+i))>>24;
                //printf("Address will be 0x%02X / %u\n", index);
                _itoa(index, szRegister, 10);
                regval = (FILTER0&valuefromfile)>>24;
                //printf("Value 0x%02X / %u\n", regval, regval);
                _itoa(regval, szByte, 10);
                DoPutReg();
                // First Byte sent

                index = (FILTER1&(address+i))>>16;
                //printf("Address will be 0x%02X / %u\n", index);
                _itoa(index, szRegister, 10);
                regval = (FILTER1&valuefromfile)>>16;
                //printf("Value 0x%02X / %u\n", regval, regval);
                _itoa(regval, szByte, 10);
                DoPutReg();
                // Second Byte sent

                index = (FILTER2&(address+i))>>8;
                //printf("Address will be 0x%02X / %u\n", index);
                _itoa(index, szRegister, 10);
                regval = (FILTER2&valuefromfile)>>8;
                //printf("Value 0x%02X / %u\n", regval, regval);
                _itoa(regval, szByte, 10);
                DoPutReg();
                // Third Byte sent

                index = FILTER3&(address+i);
                //printf("Address will be 0x%02X / %u\n", index);
                _itoa(index, szRegister, 10);
                regval = FILTER3&valuefromfile;
                //printf("Value 0x%02X / %u\n", regval, regval);
                _itoa(regval, szByte, 10);
                DoPutReg();
                // Fourth Byte sent

                i+=WORDBYTES;
                j++;
                valuefromfile = valuefromfileahead;
      //charge unsent value to be sent in next cycle
                //printf("Enter anything: \n");
                //scanf("%c\n",&temp);
                if (flag==1) break;
                complete = (j/(float)count)*100;
                printf("\r %03.1f %%", complete);
           }

      QueryPerformanceCounter(&endtick);

      timeelapsed          =          (double)(endtick.QuadPart          -
starttick.QuadPart)/freq.QuadPart;

      }
      else{
```

```
            printf("File was empty, nothing to send.\n");
            goto lFail;
      }


      if (j!=count) {
            printf("End Of File: operation completed for only %d
values\n", j);
            count = j;
      }
      else printf("Values successfully sent to the Atlys board.\n");

      printf("Values were transfered at %.1f Bytes per second.\n",
double((count*4)/timeelapsed));

lFail:
      if (filein != NULL) {
            fclose(filein);
      }

}

/*********************************************************************
/

void GetDataFromAtlys() {

      FILE *fileout;
      unsigned int regval, index, address, count, i, valuefromfpga =
0x00;
      int ret, j;
      char temp;
      float complete, timeelapsed;
      LARGE_INTEGER freq, starttick, endtick;

      if((fileout = fopen(szFile,"w"))==NULL)
      {
            printf("Error opening input file!\n");
            goto lFail;
      }

      address = atoi(szRegister);                                  //read
from stdin, address is the point at which data will start to be read
      count = atoi(szByte);                                        //read
from stdin, count is the ammount of 32 bit words that will be read to
dataFROMatlys.txt
      if ((address+(count*4))>=FLASHSIZE) {
            printf("Considering both address and data amount you'd be
addressing outside the Flash area\n");
            count = FLASHSIZE-address-1;
            printf("To avoid this data amount will be cropped to %d.\n",
count);
      }
      printf("Reading your values from Atlys Flash starting from address
0x%08X, an amount of %d words will be read to the value list.\n",
address, count);
      printf("Please hold...\n");

      /* Receive data from FPGA */
```

```
	i=0;
	j=0;

	QueryPerformanceFrequency(&freq);
	QueryPerformanceCounter(&starttick);

	while (i<(WORDBYTES*count)){

		//printf("Address will be 0x%08X / %u\n", address+i,
address+i);

		index = (FILTER0&(address+i))>>24;
		_itoa(index, szRegister, 10);
		DoGetReg();                                          // First
Byte of address word sent
		//printf("Address will be 0x%02X / %u\n", index, index);

		index = (FILTER1&(address+i))>>16;
		_itoa(index, szRegister, 10);
		DoGetReg();                                          //
Second Byte of address word sent
		//printf("Address will be 0x%02X / %u\n", index, index);

		index = (FILTER2&(address+i))>>8;
		_itoa(index, szRegister, 10);
		DoGetReg();                                          // Third
Byte of address word sent
		//printf("Address will be 0x%02X / %u\n", index, index);

		index = FILTER3&(address+i);
		_itoa(i, szRegister, 10);
		DoGetReg();                                          //
Fourth Byte of address word sent
		//printf("Address will be 0x%02X / %u\n", index, index);
		regval = (idData<<24)&FILTER0;                 //  First  Byte
of data word received
		//printf("Value 0x%08X / %u\n", regval, regval);

		DoGetReg();                                          //
Second Byte of data word received
		regval += (idData<<16)&FILTER1;
		//printf("Value 0x%08X / %u\n", regval, regval);

		DoGetReg();                                          // Third
Byte of data word received
		regval += (idData<<8)&FILTER2;
		//printf("Value 0x%08X / %u\n", regval, regval);

		DoGetReg();                                          //
Fourth Byte of data word received
		valuefromfpga = regval + (idData&FILTER3);
		//printf("Value from fpga is 0x%08X / %u\n", valuefromfpga,
valuefromfpga);

		if((ret=(fprintf(fileout,"%u\n",valuefromfpga)))<1) {
			printf("Erroneous value entered, operation failed.\n");
			goto lFail;
		}
```

```
            i+=WORDBYTES;
            j++;
            //printf("Enter anything: \n");
            //scanf("%c\n",&temp);
            complete = (j/(float)count)*100;
            printf("\r %03.1f %%", complete);
      }

      QueryPerformanceCounter(&endtick);

      timeelapsed          =          (double)(endtick.QuadPart          -
starttick.QuadPart)/freq.QuadPart;

      printf("Values successfully received and written to the output file
of your choosing.\n");

      printf("Values   were   read   at   %.1f   Bytes   per   second.\n",
double((count*4)/timeelapsed));

lFail:
      if (fileout != NULL) {
            fclose(fileout);
      }

}
```

ANEXO R

Projeto no tipo hierárquico para Atlys:

comunicações com todas as memórias externas,

escrita e leitura indireta na QuadSPI-Flash

```
/*
 *
 * Author
 * Pedro Soares
 * Universidade de Aveiro - DETI
 * pedrosoares@ua.pt
 * 27987
 *
 */

// DDR defines
#define DDRBASEADDRESS              XPAR_MCB_DDR2_MPMC_BASEADDR
#define DDRHIGHADDRESS              XPAR_MCB_DDR2_MPMC_HIGHADDR
#define MAXDDRADDRESS               0x07FFFFFF
#define     FLASHBACKUPBASEADDRESS  0x07000000
#define     OCCUPIEDMAPBASEADDRESS  0x06FF0000
#define CHANGEMAPBASEADDRESS  0x06FEFF00
#define ALLBACKUPAREA               0x01010100
#define     FLASHBKUPAREA                 0x01000000
#define     PAGEMAPAREA                   0x00010000
#define SECTORAREA                        0x00000100
// QuadSPI Flash defines
#define FLASHBASEADDRESS            XPAR_DIGILENT_QUADSPI_CNTLR_BASEADDR
#define FLASHHIGHADDRESS            XPAR_DIGILENT_QUADSPI_CNTLR_HIGHADDR
#define MAXFLASHADDRESS             0x00FFFFFF
#define FLASHSECTORSCOUNT           256
#define FLASHPAGESCOUNT             65536
#define PAGESIZE                    256
#define     SECTORSIZE                    65536
#define BYTESPERPAGE                256
#define PAGESPERSECTOR              256
#define WORD32PERSECTOR             2048
#define SUBSECTORPAGES              16
// Byte Filters
#define FILTER0                           0xFF000000
#define FILTER1                           0x00FF0000
#define FILTER2                           0x0000FF00
#define FILTER3                           0x000000FF
// Mode defines
#define QUAD                              0x00000002
#define DUAL                              0x00000001
#define EXTENDED                    0x00000000

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "xparameters.h"
#include "xenv_standalone.h"
```

```
#include "xbasic_types.h"
#include "xuartns550_l.h"
#include "usb_epp.h"
#include "xintc.h"
#include "xio.h"

//External Function Prototypes
/* LOW LEVEL - Xilinx? */
Xuint32 Read_Identification (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE,
Xuint32 CURRENT_MODE, u8 *MANUFACT_ID, u8 *MEM_TYPE, u8 *MEM_CAPACITY, u8
*EDID);
Xuint32 Fast_Read (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE, Xuint32
CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS, Xuint32 NR_OF_BYTES, Xuint32
NR_OF_DUMMY_CLKS, u8 DATA[256]);
Xuint32 Page_Program (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE, Xuint32
CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS, Xuint32 NR_OF_BYTES, u8
DATA[256]);
Xuint32 Setor_Erase (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE, Xuint32
CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS);
Xuint32 Write_Enable (Xuint32 QuadSPI_Baseaddr, Xuint32 DIV_RATE, Xuint32
MODE);
Xuint32      Write_Volatile_Enhanced_Configuration_Register      (Xuint32
QuadSPI_Baseaddr, Xuint32 DIV_RATE, Xuint32 MODE, u8 DATA);
Xuint32  Subsetor_Erase  (Xuint32  QuadSPI_Baseaddr,  Xuint32  DIV_RATE,
Xuint32 CURRENT_MODE, Xuint32 MODE, Xuint32 ADDRESS);

/* HIGH LEVEL - Digilent? */
Xuint32 Check_Initial_Mode (Xuint32 QuadSPI_Baseaddr);
Xuint32 Quad_SPI_Flash_Test (Xuint32 QuadSPI_Baseaddr);
Xuint32 Blank_Check_Entire_Memory (Xuint32 QuadSPI_Baseaddr);
Xuint32 Erase_Entire_Memory (Xuint32 QuadSPI_Baseaddr);
Xuint32 Blank_Check_Setor (Xuint32 QuadSPI_Baseaddr, Xuint32 ADDRESS);
Xuint32 Erase_Setor (Xuint32 QuadSPI_Baseaddr, Xuint32 ADDRESS);
Xuint32 Manufact_ID (Xuint32 QuadSPI_Baseaddr);
Xuint32   memory_test   (unsigned   int   MEM_BASEADDR,   unsigned   int
MEM_HIGHADDR, unsigned int DATA_WIDTH);
Xuint32 Clear_Flags (Xuint32 QuadSPI_Baseaddr);

//Internal Function Prototypes
void intro(void);
void printmenu(void);
u8 operchoice(void);
Xuint32 verbochoice(void);
Xuint32 modeselect(void);
Xuint32 memorychoice(void);
int addresSselect(Xuint32 *addr);
Xuint8 simpleHex2Dec(char hexin);
int dataCountEntry(Xuint32 *count);
int byteRead(Xuint32 *bytetoread);

//External variable definition
      //verbose char
      unsigned char verbose;

      //interrupt controller
      XIntc Intc;

      //word completion counter
      int counter = 0;
```

```
//void writetoDDR(Xuint32 datain, Xuint32 address);
//Xuint32 readfromDDR(Xuint32 address);

void usb_epp_interrupt_handler(void * baseaddr_p) {
      Xuint32 Usb_epp_data;
      Xuint32 Usb_epp_address;
      Xuint32 Usb_epp_status;
      Xuint32 Usb_temp_data;
      Xuint32 Usb_temp_address;

      Usb_epp_status    =    XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR    +
USB_EPP_STATUS_REG_OFFSET);
      //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",   Usb_epp_status);

      if (Usb_epp_status & USB_EPP_STATUS_READ_REQUESTED)
      {
            //read first the address from where read is requested
            Usb_epp_address = XIo_In32 (XPAR_DIGILENT_USB_EPP_BASEADDR +
USB_EPP_ADDRESS_REG_OFFSET);

            //filter relevant bytes of both data and address
            switch (counter) {
            case 0:
                  Usb_temp_address = (Usb_epp_address<<24)&FILTER0;
      //obtain Upper Byte of address
                  counter++;
                  break;
            case 1:
                  Usb_temp_address += (Usb_epp_address<<16)&FILTER1;
      //obtain Upper Byte of address
                  counter++;
                  break;
            case 2:
                  Usb_temp_address += ((Usb_epp_address<<8)&FILTER2);
      //obtain Middle Byte of address
                  counter++;
                  break;
            case 3:
                  Usb_temp_address += (Usb_epp_address&FILTER3);
      //obtain Base Byte of address, 3 Byte address completed
                  counter++;
                  Usb_temp_data      =      XIo_In32(DDRBASEADDRESS      +
Usb_temp_address);      //pull  4  Byte  Data  Word  from  DDR  using  4  Byte
address word
                  Usb_epp_data = (Usb_temp_data&FILTER0)>>24;
            //send Upper Byte of data word
                  break;
            case 4:
                  Usb_epp_data = (Usb_temp_data&FILTER1)>>16;
            //send Middle Byte 1 of data word
                  counter++;
                  break;
            case 5:
                  Usb_epp_data = (Usb_temp_data&FILTER2)>>8;
            //send Middle Byte 2 of data word
                  counter++;
                  break;
            case 6:
```

```
                  Usb_epp_data = Usb_temp_data&FILTER3;
            //send Base Byte of data word, data word sent
                  counter=0;
                  break;
            }

            //acknowledge the interrupt by writing into the specified
address
            XIo_Out32(XPAR_DIGILENT_USB_EPP_BASEADDR                    +
USB_EPP_DATA_REG_OFFSET, Usb_epp_data);

            //xil_printf("Data Read Performed, target address was 0x%08X
and data read back was 0x%08X\r\n", Usb_epp_address, Usb_epp_data);
      }

      else if (Usb_epp_status & USB_EPP_STATUS_WRITE_PERFORMED)
      {
            //read first the address from where write was performed
            Usb_epp_address = XIo_In32 (XPAR_DIGILENT_USB_EPP_BASEADDR +
USB_EPP_ADDRESS_REG_OFFSET);
            //acknowledge the interrupt by reading from the current
address
            Usb_epp_data   =   XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR   +
USB_EPP_DATA_REG_OFFSET);

            //shift and filter relevant data byte of both data and
address, store when entire word has been received
            switch (counter) {
            case 0:
                  Usb_temp_data           = (Usb_epp_data<<24)&FILTER0;
                  //obtain upper byte of data word
                  Usb_temp_address  = (Usb_epp_address<<24)&FILTER0;
            //obtain upper byte of address word
                  counter++;
                  break;
            case 1:
                  Usb_temp_data           +=   (Usb_epp_data<<16)&FILTER1;
                        //obtain middle byte 1 of data word
                  Usb_temp_address  += (Usb_epp_address<<16)&FILTER1;
            //obtain middle byte 1 of address word
                  counter++;
                  break;
            case 2:
                  Usb_temp_data           += (Usb_epp_data<<8)&FILTER2;
                  //obtain middle byte 2 of data word
                  Usb_temp_address  += (Usb_epp_address<<8)&FILTER2;
            //obtain middle byte 2 of address word
                  counter++;
                  break;
            case 3:
                  Usb_temp_data           += Usb_epp_data&FILTER3;
                        //obtain  base  byte  of  data  word,  data  word
complete
                  Usb_temp_address  += Usb_epp_address&FILTER3;
                  //obtain  base  byte  of  address  word,  address  word
complete
                  XIo_Out32(DDRBASEADDRESS      +      Usb_temp_address,
Usb_temp_data);  //push  to  DDR  the  complete  data  word,  using  the
complete address word
```

```
                if (Usb_temp_address > FLASHBACKUPBASEADDRESS) {
                //update Setor Map and Page Map if required (QuadSPI
Flash addressed)
                        XIo_Out8(DDRBASEADDRESS + CHANGEMAPBASEADDRESS +
((Usb_temp_address - FLASHBACKUPBASEADDRESS) / SECTORSIZE), 0x01);     //
Setor now has new info, Update Setor Map
                        XIo_Out8(DDRBASEADDRESS + OCCUPIEDMAPBASEADDRESS
+ ((Usb_temp_address - FLASHBACKUPBASEADDRESS) / PAGESIZE), 0x01);     //
Page has new info, Update Page Occupancy Map
                }
                counter=0;
                break;
           }

           //xil_printf("Data Write Performed, target address was 0x%08X
and data written was 0x%08X\r\n", Usb_epp_address, Usb_epp_data);
      }
      //rerread the status of the USB-EPP interface
      Usb_epp_status    =    XIo_In32(XPAR_DIGILENT_USB_EPP_BASEADDR    +
USB_EPP_STATUS_REG_OFFSET);
    //xil_printf("\r\nUSB_EPP_Status = 0x%0.8X",     Usb_epp_status);
}

int main()
{
//Xuint32 Usb_epp_status;

      XStatus     Status;
      Xuint32 ret;
      Xuint32 address = 0x0;
      u8          oper;
      Xuint8      destination;
      Xuint32     curvalue;
      Xuint32     curaddress;
      int   i;
      int   k;
      int   j;
      int         newj;
      Xuint32 count;
      u8          data[256], progdata1[256], progdata2[256];
      Xuint32 nrofdummyclks = 10;
      Xuint32 mode;
      u8 temp;
      Xuint32 Cur_Mode;

      /* Initialize RS232 - Set baudrate and number of stop bits */
      XUartNs550_SetBaud(XPAR_RS232_UART_1_BASEADDR,
XPAR_RS232_UART_1_CLOCK_FREQ_HZ, 115200);
      XUartNs550_SetLineControlReg(XPAR_RS232_UART_1_BASEADDR,
XUN_LCR_8_DATA_BITS);

      XCACHE_ENABLE_ICACHE();
      XCACHE_ENABLE_DCACHE();

      // Introduce project
      intro();

      //initialize interrupt controller
      Status = XIntc_Initialize (&Intc, XPAR_XPS_INTC_0_DEVICE_ID);
```

```
    if (Status != XST_SUCCESS) xil_printf ("\r\nInterrupt controller
initialization failure");
    else xil_printf("\r\nInterrupt controller initialized");

    //register usb_epp_interrupt_handler() for the system
    Status              =              XIntc_Connect              (&Intc,
XPAR_XPS_INTC_0_DIGILENT_USB_EPP_IRQ_EPP_INTR,        (XInterruptHandler)
usb_epp_interrupt_handler, (void *)XPAR_DIGILENT_USB_EPP_BASEADDR);

    if (Status != XST_SUCCESS) xil_printf ("\r\nRegistering USB_EPP
Interrupt Failed");
    else xil_printf("\r\nUSB_EPP Interrupt registered");

    //enable interrupt controller
    XIntc_Enable  (&Intc, XPAR_XPS_INTC_0_DIGILENT_USB_EPP_IRQ_EPP_INTR);
    xil_printf("\r\nInterrupt controller enabled");

    //start the interrupt controller in real mode
    Status = XIntc_Start(&Intc, XIN_REAL_MODE);
    if (Status != XST_SUCCESS) xil_printf ("\r\nInterupt controller
starting failed");
    else xil_printf("\r\nInterrupt controller started");

    //enable interrupts for the processor
    microblaze_enable_interrupts();

    //Choosing transmission rate option
    ret=modeselect();
    if (ret<0){
      xil_printf("App will now close!\r\n");
      return -1;
    }
    else mode=ret;

    // Verbose option
    ret=verbochoice();
    if (ret<0){
      xil_printf("App will now close!\r\n");
      return -1;
    }
    else verbose=ret;

    Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
    if (verbose) xil_printf("\r\n\r\nQuadSPI Flash current mode is:
%d\r\n",Cur_Mode);
    if (verbose) xil_printf("2:Quad ; 1:Dual ; 0:Extended\r\n\r\n");

      if (Cur_Mode != mode) {
            if (mode == 2) {
                  if   (verbose)   xil_printf("Clearing   QuadSPI   Flash
Flags\r\n");
                  Clear_Flags (FLASHBASEADDRESS);
                  if   (verbose)   xil_printf("\r\n\r\nWriting   Volatile
Enhanced Configuration Register\r\n");
                  Write_Volatile_Enhanced_Configuration_Register
(FLASHBASEADDRESS, 2, Cur_Mode, 0x7F); // quad command input
                  Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
                  if (verbose) xil_printf("\r\nNew mode set as: %d\r\n",
Cur_Mode);
```

```
                  if    (verbose)    xil_printf("2:Quad    ;    1:Dual    ;
0:Extended\r\n\r\n");
            }
            else if (mode == 1) {
                  if   (verbose)   xil_printf("Clearing   QuadSPI   Flash
Flags\r\n");
                  Clear_Flags (FLASHBASEADDRESS);
                  if   (verbose)   xil_printf("\r\n\r\nWriting   Volatile
Enhanced Configuration Register\r\n");
                  Write_Volatile_Enhanced_Configuration_Register
(FLASHBASEADDRESS, 2, Cur_Mode, 0xBF); // dual command input
                  Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
                  if (verbose) xil_printf("\r\nNew mode set as: %d\r\n",
Cur_Mode);
                  if    (verbose)    xil_printf("2:Quad    ;    1:Dual    ;
0:Extended\r\n\r\n");
            }
            else {
                  if   (verbose)   xil_printf("Clearing   QuadSPI   Flash
Flags\r\n");
                  Clear_Flags (FLASHBASEADDRESS);
                  if   (verbose)   xil_printf("\r\n\r\nWriting   Volatile
Enhanced Configuration Register\r\n");
                  Write_Volatile_Enhanced_Configuration_Register
(FLASHBASEADDRESS, 2, Cur_Mode, 0xFF); // single command input
                  Cur_Mode = Check_Initial_Mode (FLASHBASEADDRESS);
                  if (verbose) xil_printf("\r\nNew mode set as: %d\r\n",
Cur_Mode);
                  if    (verbose)    xil_printf("2:Quad    ;    1:Dual    ;
0:Extended\r\n\r\n");
            }
      }

    //startup Flash Backup, Page Occupancy Map and Setor Change Map
      if (verbose) xil_printf("\r\nThe contents of the QuadSPI Flash will
now be backed-up to an area of the Atlys's DDR;\r\n");
      if (verbose) xil_printf("A Page Occupancy Map will also be started
in the Atlys's DDR, reflecting non-zero pages;\r\n");
      if (verbose) xil_printf("Finally, a Changed Setor Map will also be
started.\r\n");
      if (verbose) xil_printf("First, off to erase the DDR area required
for these OPs\r\n");
    for (i=0;i<ALLBACKUPAREA;i=i+4){
                        //This process already sets Setor Change Map
      XIo_Out32(DDRBASEADDRESS + CHANGEMAPBASEADDRESS + i, 0x00000000);
    }
    if (verbose) xil_printf("Done!\r\n");
    if  (verbose)  xil_printf("\r\nReading  data  from  QuadSPI  Flash  to
DDR\r\n");
    xil_printf("Please hold...\r\n");
    for (i=0;i<FLASHPAGESCOUNT;i++){
                  //cover the entire QuadSPI Flash
      Fast_Read  (FLASHBASEADDRESS,  2,  mode,  mode,  (i  *  BYTESPERPAGE),
BYTESPERPAGE, nrofdummyclks, data);      //actually   read   data   from
QuadSPI Flash
      for(k=0;k<BYTESPERPAGE;k++){
            XIo_Out8(DDRBASEADDRESS  +  FLASHBACKUPBASEADDRESS  +  (i  *
BYTESPERPAGE) + k, data[k]);  //place read byte into DDR Flash BkUp Area
```

```
            if      (data[k]!=0xFF)       XIo_Out8(DDRBASEADDRESS      +
OCCUPIEDMAPBASEADDRESS + i, 0x1);              //update Page Ocupancy Map
as necessary
      }
    }
    xil_printf("\r\nAll operations completed successfully\r\n");

      //infinite loop, do nothing and wait for interrupts from the USB-
EPP core
    while (1)
    {
      //present options
      printmenu();
      //read option
      oper = operchoice();
      //Switch to perform desired operation
      switch (oper) {

      case 'a':                                  //Store  Data  In  QuadSPI
Flash (post USB operations)        DONE C AND TESTED -> PASSED
            for (i=0;i<FLASHSECTORSCOUNT;i++) {
                                          //Check  for  Sectors  that  have
changed
                  temp = XIo_In8(DDRBASEADDRESS + CHANGEMAPBASEADDRESS +
i);                       //Read from Setor Map
                  if (temp == 0x01){                  // When one is found
                        ret        =        Erase_Setor(FLASHBASEADDRESS,
i*SECTORSIZE);                            //  Erase  Setor  in  order  to
ensure a correct write operation
                        if (ret != 0x0){
                              xil_printf("Setor  Erase  failed  with  error
code 0x%08x.\r\n", ret);
                              goto storexit;
                        }
                        xil_printf("\r\n\r\nSector Erase for setor nr %d
completed successfully\r\n", i);
                        for (k=0;k<PAGESPERSECTOR;k++) {
                                          //Cover all Pages in a Setor
                              for (j=0;j<BYTESPERPAGE;j++) {
                                                //Cover  all  Bytes  in  a
Page
                                    if (j<128) {
                                                      //Read    to    array
values stored in Flash Backup
                                          progdata1[j]                    =
XIo_In8(DDRBASEADDRESS   +   FLASHBACKUPBASEADDRESS   +   (i*SECTORSIZE)   +
(k*PAGESIZE) + j);
                                    }

                                    else {
                                          newj = j-128;
                                          progdata2[newj]                 =
XIo_In8(DDRBASEADDRESS   +   FLASHBACKUPBASEADDRESS   +   (i*SECTORSIZE)   +
(k*PAGESIZE) + j);
                                    }
                              }
                              ret   =   Page_Program(FLASHBASEADDRESS,   2,
mode, mode, (i*SECTORSIZE) + (k*PAGESIZE), 128, progdata1);      //Page
Program of array with values from Flash Backup
```

```
                              if (ret != 0x03) {
                                    xil_printf("Page Program failed with
error code 0x%08x while writing Page %d in Setor %d.\r\n", ret, k, i);
                                    goto storexit;
                              }
                              ret = Page_Program(FLASHBASEADDRESS, 2,
mode, mode, (i*SECTORSIZE) + (k*PAGESIZE) + 128, 128, progdata2);
      //Page Program of array with values from Flash Backup
                              if (ret != 0x03) {
                                    xil_printf("Page Program failed with
error code 0x%08x while writing Page %d in Setor %d.\r\n", ret, k, i);
                                    goto storexit;
                              }
                        }
                        XIo_Out8(DDRBASEADDRESS + CHANGEMAPBASEADDRESS +
i, 0x0);                      //Setor Change Map update
                  }
            }
            xil_printf("\r\nAll data in DDR Backup updated to QuadSPI
Flash successfully\r\n");
storexit:   break;
      //-------------------------------------------------------------------
-----------------------------------------------------


      case 'b':                                 //Retrieve Data From
QuadSPI Flash (refreshing Flash Backup)       DONE C AND TESTED ->
PASSED
            if (verbose) xil_printf("\r\nThe contents of the QuadSPI
Flash will now be backed-up to an area of the Atlys's DDR;\r\n");
            if (verbose) xil_printf("Page Occupancy Map and Setor Map
will be updated;\r\n");
                  if (verbose) xil_printf("\r\nReading data from QuadSPI
Flash to DDR\r\n");
                  xil_printf("Please hold...\r\n");
            for (i=0;i<FLASHPAGESCOUNT;i++){
                                                      //read every page
from Flash
                  Fast_Read (FLASHBASEADDRESS, 2, mode, mode, i*PAGESIZE,
BYTESPERPAGE, nrofdummyclks, data);
                  for(k=0;k<BYTESPERPAGE;k++){
                        XIo_Out8(DDRBASEADDRESS + FLASHBACKUPBASEADDRESS
+ (i * BYTESPERPAGE) + k, data[k]); //place read byte into DDR Flash BkUp
Area
                        if (data[k]!=0xFF) XIo_Out8(DDRBASEADDRESS +
OCCUPIEDMAPBASEADDRESS + i, 0x1);            //update Page Ocupancy Map
as necessary
                  }
            }
            for (i=0;i<FLASHSECTORSCOUNT;i=i+4){
                                                            //Fast Reset
Setor Map
                  XIo_Out32(DDRBASEADDRESS + CHANGEMAPBASEADDRESS + i,
0x00000000);
            }
            xil_printf("QuadSPI Flash read to DDR Backup Area
successful\r\n");
            break;
```

```
      //-------------------------------------------------------------------
-------------------------------------------------------


      case 'c':                                   //Read Flash Occupancy Map
                                            DONE C AND TESTED -> PASSED
            curaddress=0;
            xil_printf("\r\nThe QuadSPI Flash is 16 MByte large; each
setor is 64 KByte large; each page 256 Byte large;\r\n");
            xil_printf("The flash occupancy is a way of identifying
available addresses in the Flash at a glance, page by page.\r\n");
            xil_printf("\r\nCoding - all '1' values report a non null
Page.\r\n");
            for (i=0;i<FLASHPAGESCOUNT;i++){   //read and print atual
flash occupancy map
                  curaddress=i*BYTESPERPAGE;
                  if ((i%64) == 0){
                        xil_printf("\r\n\r\n   -*-   Press   any   key   to
continue -*- \r\n");
                        temp                                              =
XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                        xil_printf("\r\nFlash Address 0x%08X    ->    ",
curaddress);
                  }
                  curvalue        =        XIo_In8(DDRBASEADDRESS        +
OCCUPIEDMAPBASEADDRESS + i);
                  xil_printf("%d ", curvalue);
            }
            xil_printf("\r\nFlash Occupancy Map Print Complete\r\n");
            break;
        //-------------------------------------------------------------------
-------------------------------------------------------


      case 'd':                                   //Read Setor Update Status
Map                                 DONE C AND TESTED -> PASSED
            xil_printf("\r\nThe QuadSPI Flash is 16 MByte large; each
setor is 64 KByte large; each page 256 Byte large\r\n");
            xil_printf("Since every erase operation, even those that
should precede every page program operation,\r\n");
            xil_printf("erases an entire setor, the setor update status
shows altered sectors ready to update.\r\n");
            xil_printf("\r\nCoding - all '1' values report a setor that
has new values yet to be written/updated to the Quad SPI Flash\r\n");
            for (i=0;i<FLASHSECTORSCOUNT;i++) {
                  curaddress=i*SECTORSIZE;
                  if ((i%16) == 0) xil_printf("\r\nFlash Address 0x%08X
->   ", curaddress);
                  curvalue        =        XIo_In8(DDRBASEADDRESS        +
CHANGEMAPBASEADDRESS + i);
                  xil_printf("%d ", curvalue);
            }
            xil_printf("\r\nSectors Update Map Print Complete\r\n");
            break;
      //-------------------------------------------------------------------
-------------------------------------------------------
```

```
      case 'e':                                    //Fast Test DDR2 Memory
                                                DONE and TESTED -> PASSED
            ret = memory_test (DDRBASEADDRESS, DDRHIGHADDRESS, 32);
      //32 is DATA_WIDTH, but it's not actually used inside function
            if (ret != 0x00) {
                  xil_printf("\r\n\r\nDDR  test  failed  with  error  at
address 0x08X.\r\n",ret);
            }
            else {
                  xil_printf("\r\n\r\nDDR          test          passed
successfully.\r\n");
                  xil_printf("This  test  however  is  destructive,  please
run operation b) to refresh the QuadSPI Flash Backup\r\n");
            }
            break;
        //-------------------------------------------------------------
-------------------------------------------------------


      case 'f':                                    //Fast  Test  QuadSPI  Flash
                                                DONE C AND TESTED -> PASSED
            ret = Quad_SPI_Flash_Test (FLASHBASEADDRESS);
            if (ret == 0x00) {
                  xil_printf("\nFlash test passed");
            }
            else if (ret == 0x80) {
                  xil_printf("\nErasing memory failed");
                  break;
            }
            else if (ret == 0x70) {
                  xil_printf("\nFlash memory not found");
                  break;
            }
            else if (ret == 0x60) {
                  xil_printf("\nWrite to Flash memory failed");
                  break;
            }
            else if (ret == 0x50) {
                  xil_printf("\nRead from Flash memory failed");
                  break;
            }
            else if (ret == 0x40) {
                  xil_printf("\nError on line DQ3");
                  break;
            }
            else if (ret == 0x30) {
                  xil_printf("\nError on line DQ2");
                  break;
            }
            else if (ret == 0x20) {
                  xil_printf("\nError on line DQ1");
                  break;
            }
            else {
                  xil_printf("\nError on line DQ0");
                  break;
            }
```

```
            xil_printf ("\r\nNow replacing data erased by test (initial
subsetor), please hold...\r\n");
            Subsetor_Erase (FLASHBASEADDRESS, 2, mode, mode, 0);
                                    //first to reerase subsetor, prepare
it for a write.
            for (i=0;i<SUBSECTORPAGES;i++){
                                                        //at this point a
subsetor (4KB) at the beggining of the QuadSPI Flash has been erased, now
to replace data erased.
                  for (j=0;j<BYTESPERPAGE;j++) {
                                                        //Cover all Bytes in
a Page
                        if (j<128) {
                                                              //Read      to
array values stored in Flash Backup
                              progdata1[j]  =  XIo_In8(DDRBASEADDRESS  +
FLASHBACKUPBASEADDRESS + (i*PAGESIZE) + j);
                        }
                        else {
                              newj = j-128;
                              progdata2[newj] = XIo_In8(DDRBASEADDRESS +
FLASHBACKUPBASEADDRESS + (i*PAGESIZE) + j);
                        }
                  }
                  ret  =  Page_Program(FLASHBASEADDRESS,  2,  mode,  mode,
i*PAGESIZE, 128, progdata1);             //Page  Program  of  array  with
values from Flash Backup
                  if (ret != 0x03) {
                        xil_printf("Page  Program  failed  with  error  code
0x%08x while writing Page %d in Setor %d.\r\n", ret, i, 0);
                        goto testexit;
                  }
                  ret  =  Page_Program(FLASHBASEADDRESS,  2,  mode,  mode,
(i*PAGESIZE) + 128, 128, progdata2);     //Page  Program  of  array  with
values from Flash Backup
                  if (ret != 0x03) {
                        xil_printf("Page  Program  failed  with  error  code
0x%08x while writing Page %d in Setor %d.\r\n", ret, i, 0);
                        goto testexit;
                  }
            }
            xil_printf("\r\nData          replacement          completed
successfully\r\n");
testexit:   break;
       //-------------------------------------------------------------------
-------------------------------------------------------------


     case 'g':                                  //QuadSPI    Flash    Setor
Erase                              DONE C AND TESTED -> PASSED
            //address read
            ret = addresSselect(&address);
            if (ret!=0x00) {
                  xil_printf("Address selection failed\r\n");
                  goto sectexit;
            }
            //erase setor from Flash
```

```
            ret = Erase_Setor(FLASHBASEADDRESS, address);
      //actually erase setor in Flash -> any address inside a setor is a
valid setor address
            if (ret!=0x00) {
                  xil_printf("Setor   Erase   failed   with   error   code
0x%08x.\r\n", ret);
                  goto sectexit;
            }
            //Erase Setor from DDR Backup
            for (i=0;i<SECTORSIZE;i=i+4) {
                  //fill  with  0xFF  a  setor  size  area  of  Flash  Backup
inside the DDR,
                  XIo_Out32(DDRBASEADDRESS   +   FLASHBACKUPBASEADDRESS   +
((address / SECTORSIZE) * SECTORSIZE) + i, 0xFFFFFFFF);
            }
            //Update Setor Map
            XIo_Out8(DDRBASEADDRESS + CHANGEMAPBASEADDRESS + (address /
SECTORSIZE), 0x00);          //fill  with  0x00  a  byte  of  Setor  Change
area in DDR, since setor has no new info and has been erased
            //Update Page Map
            for (i=0;i<PAGESPERSECTOR;i=i+4) {
                        //fill  with  0x00  a  setor  size  area  of  Page
Occupancy inside the DDR, since pages are effectively blank
                  XIo_Out32(DDRBASEADDRESS   +   OCCUPIEDMAPBASEADDRESS   +
(((address / SECTORSIZE) * SECTORSIZE) / PAGESIZE) + i, 0x00000000);
            }
            xil_printf("Setor Erase OPs successfully completed\r\n");
sectexit:   break;
        //-------------------------------------------------------------------
-----------------------------------------------------------


      case 'h':                                    //QuadSPI    Flash    Erase
Memory                                    DONE C AND TESTED -> PASSED
            xil_printf("\r\nNow erasing, please wait...\r\n");
            ret = Erase_Entire_Memory (FLASHBASEADDRESS);
            if (ret!=0x00) {
                  xil_printf("Erase  Setor  failed,  error  was  0x%x\r\n",
ret);
                  goto erasexit;
            }
            for (i=0;i<(PAGEMAPAREA + SECTORAREA);i=i+4){
                  // Clear both Setor Change and Page Occupancy Map
                  XIo_Out32(DDRBASEADDRESS  +  CHANGEMAPBASEADDRESS  +  i,
0x00000000);
            }
            for  (i=0;i<((MAXDDRADDRESS-FLASHBACKUPBASEADDRESS)+1);i=i+4){
                                    // Clear Flash Backup
                  XIo_Out32(DDRBASEADDRESS + FLASHBACKUPBASEADDRESS + i,
0xFFFFFFFF);
            }
            xil_printf("\r\nQuadSPI  Flash  Memory  Erase  OPs  successfully
completed\r\n");
erasexit:   break;
        //-------------------------------------------------------------------
-----------------------------------------------------------
```

```
      case 'i':                                  //Generate Random Data And
Fill Choice Memory           DONE C
            //destination
            ret = memorychoice();
            if (ret < 0x00) {
                  xil_printf("Memory choice failed, try again\r\n");
                  goto genexit;
            }
            else destination = ret;
            //address read
            ret = addresSselect(&address);
            if (ret!=0x00) {
                  xil_printf("Address selection failed\r\n");
                  goto genexit;
            }
            //data count read
            ret = dataCountEntry(&count);
            if (ret!=0x00) {
                  xil_printf("Data amount selection failed\r\n");
                  goto genexit;
            }
            //data generation and placement
            rand();
      //compatibility startup
            srand((Xuint32)time(NULL));                     //random  seed
setup
            switch (destination){
            case 1:                                                 //   DDR
selected
                  if (address > CHANGEMAPBASEADDRESS) {
                        xil_printf("Operation would start at a reserved
DDR memory area\r\n");
                        xil_printf("Thus it will now break\r\n");
                        goto genexit;
                  }
                  if ((address + count) > CHANGEMAPBASEADDRESS) {
                        xil_printf("Operation partly addresses a reserved
DDR memory area\r\n");
                     xil_printf("Thus the amount of data required cannot
be fulfilled\r\n");
                     count = (CHANGEMAPBASEADDRESS - address);
                     xil_printf("For this opp, only 0x%08X bytes will be
genereated and placed\r\n", count);
                  }
                  xil_printf("\r\nNow  filling  selected  memory.  Please
hold...\r\n");
                  for (i=0;i<count;i++){                        //random
filler generation
                        curvalue                                        =
(u8)((char)((int)((double)254*rand()/(RAND_MAX+1.0))));
                        XIo_Out8(DDRBASEADDRESS + address + i, curvalue);
                  }
                  xil_printf("Data     generation     and     placement
complete.\r\n");
                  break;
            case 0:                                                 // Flash
selected
                  if (address > MAXFLASHADDRESS) {
```

```
                        xil_printf("Operation would address over the
QuadSPI Flash size\r\n");
                        xil_printf("Thus it will now break\r\n");
                        goto genexit;
                }
                if ((address + count) > MAXFLASHADDRESS) {
                        xil_printf("Operation partly addresses over the
QuadSPI Flash size\r\n");
                        xil_printf("Thus the amount of data required
cannot be fulfilled\r\n");
                        count = ((MAXFLASHADDRESS + 1) - address);
                        xil_printf("For this opp, only 0x%08X bytes will
be genereated and placed\r\n", count);
                }
                xil_printf("\r\nNow filling selected memory. Please
hold...\r\n");
                for (i=0;i<count;i++){                          //random
filler generation
                        curvalue                                        =
(u8)((char)((int)((double)254*rand()/(RAND_MAX+1.0))));
                        XIo_Out8(DDRBASEADDRESS + FLASHBACKUPBASEADDRESS
+ address + i, curvalue);               // Place new data into position
on Flash Backup Area
                        XIo_Out8(DDRBASEADDRESS + CHANGEMAPBASEADDRESS +
((address + i)/SECTORSIZE), 0x01); // Setor now has new info, Update
Setor Map
                        XIo_Out8(DDRBASEADDRESS + OCCUPIEDMAPBASEADDRESS
+ ((address + i)/PAGESIZE), 0x01); // Page has new info, Update Page
Occupancy Map
                }
                xil_printf("Data     generation     and     placement
complete.\r\n");
                xil_printf("Don't forget to run operation a) to
effectively store new values to QuadSPI Flash.\r\n");
                break;
        }
genexit:    break;
        //-------------------------------------------------------------
-------------------------------------------------------------


    case 'j':                                   //Read Choice Memory
                                DONE C AND TESTED -> PASSED
        //destination
        ret = memorychoice();
        if (ret < 0x00) {
                xil_printf("Memory choice failed, try again\r\n");
                goto readexit;
        }
        else destination = ret;
        //address read
        ret = addresSselect(&address);
        if (ret!=0x00) {
                xil_printf("Address selection failed\r\n");
                goto readexit;
        }
        //data count read
        ret = dataCountEntry(&count);
        if (ret!=0x00) {
```

```
                xil_printf("Data amount selection failed\r\n");
                goto readexit;
          }
          switch (destination){
          case 1:                                                    //   DDR
selected
                if (address > CHANGEMAPBASEADDRESS) {
                      xil_printf("Operation would start at a reserved
DDR memory area\r\n");
                      xil_printf("Thus it will now break\r\n");
                      goto readexit;
                }
                if ((address + count) > CHANGEMAPBASEADDRESS) {
                      xil_printf("Operation partly addresses a reserved
DDR memory area\r\n");
                      xil_printf("Thus  the  amount  of  data  required
cannot be fulfilled\r\n");
                      count = (CHANGEMAPBASEADDRESS - address);
                      xil_printf("For this opp, only 0x%08X bytes will
be read\r\n", count);
                }
                for (i=0;i<count;i++){
                      curaddress = address + i;
                      if  ((i%32)  ==  0)  xil_printf("\r\nDDR  Address
0x%08X ", curaddress);
                      curvalue = XIo_In8(DDRBASEADDRESS + address + i);
                      xil_printf("0x%02X ", curvalue);
                }
                xil_printf("\r\n\r\nData read successfully.\r\n");
                break;
          case 0:                                                    // Flash
selected
                if (address > MAXFLASHADDRESS) {
                      xil_printf("Operation  would  address  over  the
QuadSPI Flash size\r\n");
                      xil_printf("Thus it will now break\r\n");
                      goto readexit;
                }
                if ((address + count) > MAXFLASHADDRESS) {
                      xil_printf("Operation  partly  addresses  over  the
QuadSPI Flash size\r\n");
                      xil_printf("Thus  the  amount  of  data  required
cannot be fulfilled\r\n");
                      count = ((MAXFLASHADDRESS + 1) - address);
                      xil_printf("For this opp, only 0x%08X bytes will
be read\r\n", count);
                }
                for (i=0;i<count;i++) {
                      curaddress = address + i;
                      if  ((i%32)  ==  0)  xil_printf("\r\nFlash  Address
0x%08X ",curaddress);
                      curvalue      =      XIo_In8(DDRBASEADDRESS      +
FLASHBACKUPBASEADDRESS + address + i);
                      xil_printf("0x%02X ", curvalue);
                }
                xil_printf("\r\n\r\nData read successfully.\r\n");
                break;
          }
readexit:break;
```

```
      //-------------------------------------------------------------------
-------------------------------------------------------


      case 'k':                                  //Write Choice Memory
                              DONE C AND TESTED -> PASSED
            //destination
            ret = memorychoice();
            if (ret < 0x00) {
                  xil_printf("Memory choice failed, try again\r\n");
                  goto writexit;
            }
            else destination = ret;
            //address read
            ret = addresSselect(&address);
            if (ret!=0x00) {
                  xil_printf("Address selection failed\r\n");
                  goto writexit;
            }
            //data count read
            ret = dataCountEntry(&count);
            if (ret!=0x00) {
                  xil_printf("Data amount selection failed\r\n");
                  goto writexit;
            }
            //data read and placement on memory
            switch (destination){
            case 1:                                                 //   DDR
selected
                  if (address > CHANGEMAPBASEADDRESS) {
                        xil_printf("Operation would start at a reserved
DDR memory area\r\n");
                        xil_printf("Thus it will now break\r\n");
                        goto writexit;
                  }
                  if ((address + count) > CHANGEMAPBASEADDRESS) {
                        xil_printf("Operation partly addresses a reserved
DDR memory area\r\n");
                        xil_printf("Thus  the  amount  of  data  required
cannot be fulfilled\r\n");
                        count = (CHANGEMAPBASEADDRESS - address);
                        xil_printf("For this opp, only 0x%08X bytes will
be written\r\n", count);
                  }
                  for (i=0;i<count;i++){
                        ret = byteRead(&curvalue);
                        if (ret != 0){
                              xil_printf("Data   value   reading   failed,
operation has failed.\r\n");
                              goto writexit;
                        }
                        XIo_Out8(DDRBASEADDRESS + address + i, curvalue);
                  }
                  xil_printf("\r\n\r\nData written successfully.\r\n");
                  break;
            case 0:                                                 // Flash
selected
                  if (address > MAXFLASHADDRESS) {
```

```
                        xil_printf("Operation would address over the
QuadSPI Flash size\r\n");
                        xil_printf("Thus it will now break\r\n");
                        goto writexit;
                  }
                  if ((address + count) > MAXFLASHADDRESS) {
                        xil_printf("Operation partly addresses over the
QuadSPI Flash size\r\n");
                        xil_printf("Thus the amount of data required
cannot be fulfilled\r\n");
                        count = ((MAXFLASHADDRESS + 1) - address);
                        xil_printf("For this opp, only 0x%08X bytes will
be written\r\n", count);
                  }
                  for (i=0;i<count;i++){
                        ret = byteRead(&curvalue);
                        if (ret != 0){
                              xil_printf("Data value reading failed,
operation has failed.\r\n");
                              goto writexit;
                        }
                        XIo_Out8(DDRBASEADDRESS + FLASHBACKUPBASEADDRESS
+ address + i, curvalue);               // Place new data into position
on Flash Backup Area
                        XIo_Out8(DDRBASEADDRESS + CHANGEMAPBASEADDRESS +
((address + i)/SECTORSIZE), 0x01); // Setor now has new info, Update
Setor Map
                        XIo_Out8(DDRBASEADDRESS + OCCUPIEDMAPBASEADDRESS
+ ((address + i)/PAGESIZE), 0x01); // Page has new info, Update Page
Occupancy Map
                  }
                  xil_printf("\r\n\r\nData written successfully.\r\n");
                  xil_printf("Don't forget to run operation a) to
effectively store new values to QuadSPI Flash.\r\n");
                  break;
            }
writexit:break;
          //-------------------------------------------------------------
-------------------------------------------------------------


      case 'm':                                 //Erase Bytes From Choice
Memory                                    DONE C AND TESTED -> PASSED
            //destination
            ret = memorychoice();
            if (ret < 0x00) {
                  xil_printf("Memory choice failed, try again\r\n");
                  goto eras2xit;
            }
            else destination = ret;
            //address read
            ret = addresSselect(&address);
            if (ret!=0x00) {
                  xil_printf("Address selection failed\r\n");
                  goto eras2xit;
            }
            //data count read
            ret = dataCountEntry(&count);
            if (ret!=0x00) {
```

```
                xil_printf("Data amount selection failed\r\n");
                goto eras2xit;
          }
          switch (destination){
          case 1:                                                  //   DDR
selected
                if (address > CHANGEMAPBASEADDRESS) {
                      xil_printf("Operation would start at a reserved
DDR memory area\r\n");
                      xil_printf("Thus it will now break\r\n");
                      goto eras2xit;
                }
                if ((address + count) > CHANGEMAPBASEADDRESS) {
                      xil_printf("Operation partly addresses a reserved
DDR memory area\r\n");
                      xil_printf("Thus the amount of data required
cannot be fulfilled\r\n");
                      count = (CHANGEMAPBASEADDRESS - address);
                      xil_printf("For this opp, only 0x%08X bytes will
be erased\r\n", count);
                }
                for (i=0;i<count;i++){
                      XIo_Out8(DDRBASEADDRESS + address + i, 0x00);
                }
                xil_printf("\r\n\r\nData erased successfully.\r\n");
                break;
          case 0:
                                              // Flash selected
                if (address > MAXFLASHADDRESS) {
                      xil_printf("Operation would address over the
QuadSPI Flash size\r\n");
                      xil_printf("Thus it will now break\r\n");
                      goto eras2xit;
                }
                if ((address + count) > MAXFLASHADDRESS) {
                      xil_printf("Operation partly addresses over the
QuadSPI Flash size\r\n");
                      xil_printf("Thus the amount of data required
cannot be fulfilled\r\n");
                      count = ((MAXFLASHADDRESS + 1) - address);
                      xil_printf("For this opp, only 0x%08X bytes will
be erased\r\n", count);
                }
                for (i=0;i<count;i++){
                      XIo_Out8(DDRBASEADDRESS + FLASHBACKUPBASEADDRESS
+ address + i, 0xFF);                          //  Erase byte of Flash
Backup Area
                      XIo_Out8(DDRBASEADDRESS + CHANGEMAPBASEADDRESS +
((address + i)/SECTORSIZE), 0x01);       //  Setor now has new info,
Update Setor Map
                }
                for (i=0;i<((count / PAGESIZE) + 1);i++) {
                                                    //  Pages have new
info, if necessary update Page Occupancy Map
                      ret = 0;
                      for (j=0;j<PAGESIZE;j=j+4) {
                            curvalue    =    XIo_In32(DDRBASEADDRESS    +
FLASHBACKUPBASEADDRESS + (((address / PAGESIZE) + i) * PAGESIZE) + j);
                            if (curvalue != 0xFFFFFFFF) ret = 1;
```

```
                        }
                        if   (ret   ==   0)   XIo_Out8(DDRBASEADDRESS   +
OCCUPIEDMAPBASEADDRESS + (address / PAGESIZE) + i, 0x00);        //In
case page is now empty, signal this emptying this cell in Page Occupancy
Map
                  }
                  xil_printf("\r\n\r\nData erased successfully.\r\n");
                  xil_printf("Don't   forget   to   run   operation   a)   to
effectively erase the values from QuadSPI Flash.\r\n");
                  break;
            }
eras2xit:   break;
      //-------------------------------------------------------------------
-------------------------------------------------------


      case 'z':                                    //TESTS
            xil_printf("\r\n\r\nNothing   to   see   here,   go   about   your
business now...\r\n\r\n");
            break;
        //-------------------------------------------------------------------
-------------------------------------------------------


      case 'q':                                       //Simplest Quit Routine
                                          DONE and TESTED -> PASSED
            goto quit;
            break;
        //-------------------------------------------------------------------
-------------------------------------------------------


      default:
            xil_printf("\n\rPressed key does not map to any available
operation, try again\r\n");
            break;
      }
    }

quit: XCACHE_DISABLE_ICACHE();
            XCACHE_DISABLE_DCACHE();
            return 0;
}

void intro(void) {
      xil_printf("\r\n\r\n\r\nComms Project - USB based\r\n");
      xil_printf("DDR2 => QuadSPI FlashData\r\n");
      xil_printf("QuadSPI FlashData => DDR2\r\n");
}


void printmenu(void) {
      xil_printf("\r\nChoose   the   desired   operation   pressing   the   key
associated with the function\r\n");
      xil_printf("!!!WARNING - To avoid data corruption, ensure that no
operations   are   working   USB   side   while   one   of   these   functions   is
running!!!\r\n");
      xil_printf("a)   Store   Data   In   QuadSPI   Flash   (post   USB
operations)\r\n");                  //DONE C and TESTED -> PASSED
```

```
	xil_printf("b)  Retrieve  Data  From  QuadSPI  Flash  (pre  USB
operations)\r\n");		//DONE C AND TESTED -> PASSED
	xil_printf("c) Read Flash Occupancy Map\r\n");
				//DONE C AND TESTED -> PASSED
	xil_printf("d) Read Setor Update Status\r\n");
				//DONE C AND TESTED -> PASSED
	xil_printf("!!!WARNING!!!  -  We  remind  you  that  you  are  not
addressing the QuadSPI Flash directly\r\n");
	xil_printf("Other Support Operations\r\n");
					//----------------------------------
--
	xil_printf("e) Fast Test DDR2 Memory\r\n");
					//DONE C AND TESTED -> PASSED
	xil_printf("f)  Fast  Test  QuadSPI  Flash  (recommended  running  a)
before)\r\n");	//DONE C AND TESTED -> PASSED
	xil_printf("g) QuadSPI Flash Setor Erase\r\n");
				//DONE C AND TESTED -> PASSED
	xil_printf("h) QuadSPI Flash Erase Memory\r\n");
				//DONE C AND TESTED -> PASSED
	xil_printf("i)  Generate  Random  Data  And  Fill  Choice  Memory\r\n");
				//DONE C
	xil_printf("j) Read Data From Choice Memory\r\n");
				//DONE C AND TESTED -> PASSED
	xil_printf("k) Write Data To Choice Memory\r\n");
				//DONE C AND TESTED -> PASSED
	xil_printf("m) Erase Data From Choice Memory\r\n");
				//DONE C AND TESTED -> PASSED
	xil_printf("z) TESTS!!!\r\n");
					//----------------------------
--------
	xil_printf("q) Quit Program\r\n");
					//DONE C AND TESTED -> PASSED
}

u8 operchoice(void) {
	u8 operchoice;

	xil_printf("\r\n  Standing by for your input    \r\n");
	operchoice = XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
	xil_printf("\r\nThe Selected Operation was: %c) \r\n",operchoice);
	return operchoice;
}

Xuint32 verbochoice(void) {
	char verbo;

	xil_printf("\r\nDo wish a VERBOSE operation? (y OR n)\r\n");
	xil_printf("      Standing by for your input    \r\n");
	verbo = XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
	if ((verbo!='y')&(verbo!='n')){
		xil_printf("\r\nThe  pressed  key  does  not  map  to  either
option, please try again\r\n");
		return -1;
	}
	else{
		if (verbo=='y') return 1;
		else return 0;
	}
```

```
}

Xuint32 modeselect(void){
	u8 newmode;

	xil_printf("\r\nChoose the transmission mode desired: Quad(q),
Dual(d) or Extended(e)\r\n");
	newmode = XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
	if (newmode=='Q')(newmode='q');
	if (newmode=='D')(newmode='d');
	if (newmode=='E')(newmode='e');
	if ((newmode!='q')&(newmode!='d')&(newmode!='e')){
		xil_printf("\r\nThe pressed key does not map to one of the
options, please try again\r\n");
		return -1;
	}
	switch (newmode){
	case 'q': return QUAD;
		break;
	case 'd': return DUAL;
		break;
	}
	return EXTENDED;
}

Xuint32 memorychoice(void) {
	u8 mem;

	xil_printf("\r\nDo wish a to address the DDR or the QuadSPI Flash?
(a OR b)\r\n");
	xil_printf("	Standing by for your input	\r\n");
	mem = XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
	if (mem == 'A') mem = 'a';
	if (mem == 'B') mem = 'b';
	if ((mem != 'a')&(mem != 'b')){
		xil_printf("\r\nThe pressed key does not map to either
option, please try again\r\n");
		return -1;
	}
	else{
		if (mem == 'a') return 1;
		else return 0;
	}
}

int addresSselect(Xuint32 *addr){
	u8 buffer[8];
	int i=0, ret;
	Xuint32 temp=0x0;

	xil_printf("\r\nChoose the starting address for this op\r\n");
	xil_printf("0x0 to 0x00FFFFFF for QuadSPI Flash Addresses\r\n");
	xil_printf("0x0 to 0x06FEFF00 for DDR2 Addresses\r\n");
	xil_printf("Terminate your option filling in the 8 hex
chars.\r\n");
	for (i=0;i<8;i++){
		buffer[i] = XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
		xil_printf("%c",buffer[i]);
	}
```

```
	i=0;
	while ((ret!=0xFF)&(i<8)){
		ret=simpleHex2Dec(buffer[i]);
		temp = (temp<<(4)) + ret;
		i+=1;
	}
	if (ret==0xFF){
		xil_printf("Bad choice of address, try again!\r\n");
		return -1;
	}
	else{
		xil_printf("\r\nYour address is setup as 0x%08x or %d in
decimal.\r\n", temp, temp);
		*addr=temp;
	}
	return 0;
}

Xuint8 simpleHex2Dec(char hexin){
	if (hexin=='0')return 0x0;
	if (hexin=='1')return 0x1;
	if (hexin=='2')return 0x2;
	if (hexin=='3')return 0x3;
	if (hexin=='4')return 0x4;
	if (hexin=='5')return 0x5;
	if (hexin=='6')return 0x6;
	if (hexin=='7')return 0x7;
	if (hexin=='8')return 0x8;
	if (hexin=='9')return 0x9;
	if (hexin=='a')return 0xa;
	if (hexin=='b')return 0xb;
	if (hexin=='c')return 0xc;
	if (hexin=='d')return 0xd;
	if (hexin=='e')return 0xe;
	if (hexin=='f')return 0xf;
	if (hexin=='A')return 0xa;
	if (hexin=='B')return 0xb;
	if (hexin=='C')return 0xc;
	if (hexin=='D')return 0xd;
	if (hexin=='E')return 0xe;
	if (hexin=='F')return 0xf;
	else return 0xFF;
}

int dataCountEntry(Xuint32 *count){
	u8 buffer[8];
	int i=0, ret;
	Xuint32 temp=0x00000000;

	xil_printf("\r\nChoose the amount of data you wish to use in this
op.\r\n");
	xil_printf("Terminate your option filling in the 8 hex chars that
represent that value.\r\n");
	for (i=0;i<8;i++){
		buffer[i] = XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
		xil_printf("%c",buffer[i]);
	}
	i=0;
	while ((ret!=0xFF)&(i<8)){
```

```
            ret=simpleHex2Dec(buffer[i]);
            temp = (temp<<(4)) + ret;
            i+=1;
      }
      if (ret==0xFF){
            xil_printf("Bad choice of data amount, try again!\r\n");
            return -1;
      }
      else{
            xil_printf("\r\nYour data amount is setup as 0x%08x or %d in
decimal.\r\n", temp, temp);
            *count=temp;
      }
      return 0;
}

int byteRead(Xuint32 *bytetoread){
      u8 buffer[2];
      int ret1, ret2;

      xil_printf("Enter your byte value now\r\n");
      buffer[1]=XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
      xil_printf(" %c", buffer[1]);
      buffer[2]=XUartNs550_RecvByte(XPAR_RS232_UART_1_BASEADDR);
      xil_printf("%c",buffer[2]);
      ret1=simpleHex2Dec(buffer[1]);
      ret2=simpleHex2Dec(buffer[2]);
      if ((ret1<0)|(ret2<0)){
            xil_printf("You've entered a char that is not an hex
value!\r\n");
            return -1;
      }
      else *bytetoread=((ret1<<4)+ret2);
      return 0;
}
```

ANEXO S

Código que implementa a aplicação de consola que permite o controlo do projeto de topo hierárquico criado para a Atlys

```
/************************************************************************
/
/*                                      */
/*  DeppDemo.cpp  --  DEPP DEMO Main Program
          */
/*                                      */
/************************************************************************
/
/*  Author: Aaron Odell
                        */
/*  Copyright:    2010 Digilent, Inc.
                        */
/************************************************************************
/
/*  Module Description:
                        */
/*          DEPP Demo demonstrates how to transfer data to and from
          */
/*          a Digilent FPGA board using the DEPP module of the Adept
     */
/*          SDK.
                            */
/*                                      */
/************************************************************************
/
/*  Revision History:
                        */
/*                                      */
/*    03/02/2010(AaronO): created
                        */
/*    07/10/2012(PedroSoares): heavily edited
                                                                        */
/************************************************************************
/
/*
*
* Heavily edited by Pedro Soares
* Universidade de Aveiro - DETI
* pedrosoares@ua.pt
* 27987
*
*/

#define     _CRT_SECURE_NO_WARNINGS
#define UPPERLIMIT      0xFFFFFFFF
#define     LOWERLIMIT  0x00000000
// Byte Filters
```

```
#define FILTER0         0xFF000000
#define FILTER1         0x00FF0000
#define FILTER2         0x0000FF00
#define FILTER3         0x000000FF

#define WORDBYTES 4

/* ------------------------------------------------------------ */
/*                        Sync With MicroBlaze Mapping
          */
/* ------------------------------------------------------------ */

// DDR defines
#define MAXDDRADDRESS               0x07FFFFFF
#define     FLASHBACKUPBASEADDRESS  0x07000000
#define     OCCUPIEDMAPBASEADDRESS  0x06FF0000
#define CHANGEMAPBASEADDRESS  0x06FEFF00
#define ALLBACKUPAREA               0x01010100

// QuadSPI Flash defines
#define MAXFLASHADDRESS             0x00FFFFFF

/* ------------------------------------------------------------ */
/*                           Include File Definitions
          */
/* ------------------------------------------------------------ */

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "dpcdecl.h"
#include "depp.h"
#include "dmgr.h"

/* ------------------------------------------------------------ */
/*                           Local Type and Constant Definitions
     */
/* ------------------------------------------------------------ */

const int cchSzLen = 1024;
const int cbBlockSize = 1000;

/* ------------------------------------------------------------ */
/*                           Global Variables
          */
/* ------------------------------------------------------------ */

BOOL            fDvc;
BOOL            fFile;
//BOOL                fFile1;
BOOL            fByte;

// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE

BOOL            fSend4BFPGA;
BOOL            fGet4BFPGA;
```

```
BOOL              fSendDataToAtlys;
BOOL              fGetDataFromAtlys;
//BOOL                  fCompareData;
//BOOL                  fGenRandData;

// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE

char              szAction[cchSzLen];
char              szRegister[cchSzLen];
char              szDvc[cchSzLen];
char              szFile[cchSzLen];
//char                  szFile1[cchSzLen];
char              szByte[cchSzLen];

HIF                   hif = hifInvalid;

FILE *                fhin = NULL;
FILE *                fhout = NULL;

// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE

unsigned int      data[4096];
unsigned int      dataFPGA[4096];
int               n=4096;

BYTE        idData;

// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE

/* ------------------------------------------------------------ */
/*                       Local Variables
                */
/* ------------------------------------------------------------ */


/* ------------------------------------------------------------ */
/*                            Forward Declarations
            */
/* ------------------------------------------------------------ */

BOOL        FParseParam(int cszArg, char * rgszArg[]);
void        ShowUsage(char * sz);
BOOL        FInit();
void        ErrorExit();

void        DoDvcTbl();
void        DoPutReg();
void        DoGetReg();

// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE

void        Send4BFPGA();
void        Get4BFPGA();
void        SendDataToAtlys();
void        GetDataFromAtlys();
//void            GenRandData();
//void            CompareData();
int             memoryops(UINT32 *address,int count);
```

```
// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE

void        StrcpyS( char* szDst, size_t cchDst, const char* szSrc );

/* ------------------------------------------------------------ */
/*                      Procedure Definitions
            */
/* ------------------------------------------------------------ */
/***  main
**
**    Synopsis
**          int main(cszArg, rgszArg)
**
**    Input:
**          cszArg            - count of command line arguments
**          rgszArg           - array of command line arguments
**
**    Output:
**          none
**
**    Errors:
**          Exits with exit code 0 if successful, else non-zero
**
**    Description:
**          main function of DEPP Demo application.
*/

int main(int cszArg, char * rgszArg[]) {

      if (!FParseParam(cszArg, rgszArg)) {
            ShowUsage(rgszArg[0]);
            return 1;
      }

      // DMGR API Call: DmgrOpen
      if(!DmgrOpen(&hif, szDvc)) {
            printf("DmgrOpen   failed   (check   the   device   name   you
provided)\n");
            return 0;
      }

      // DEPP API call: DeppEnable
      if(!DeppEnable(hif)) {
            printf("DeppEnable failed\n");
            return 0;
      }

      else if (fSendDataToAtlys) {
            SendDataToAtlys();
      }

      else if (fGetDataFromAtlys) {
            GetDataFromAtlys();
      }

      else if (fSend4BFPGA) {
            Send4BFPGA();
      }
```

```
      else if (fGet4BFPGA) {
            Get4BFPGA();
      }

      /*else if (fCompareData) {
            CompareData();
      }*/

      /*else if (fGenRandData) {
            GenRandData();
      }*/

      if( hif != hifInvalid ) {
            // DEPP API Call: DeppDisable
            DeppDisable(hif);

            // DMGR API Call: DmgrClose
            DmgrClose(hif);
      }

      return 0;
}

/* ------------------------------------------------------------ */
/***  DoGetReg
**
**    Synopsis
**          void DoGetReg()
**
**    Input:
**          none
**
**    Output:
**          none
**
**    Errors:
**          none
**
**    Description:
**          Gets a byte from, a specified register
*/

void DoGetReg() {

      BYTE  idReg;
      char *      szStop;

      idReg = (BYTE) strtol(szRegister, &szStop, 10);

      // DEPP API Call: DeppGetReg
      if(!DeppGetReg(hif, idReg, &idData, fFalse)) {
            printf("DeppGetReg failed\n");
            ErrorExit();
      }

      return;
}

/* ------------------------------------------------------------ */
```

```
/***  DoPutReg
**
**    Synopsis
**          void DoPutReg()
**
**    Input:
**          none
**
**    Output:
**          none
**
**    Errors:
**          none
**
**    Description:
**          Sends a byte to a specified register
*/

void DoPutReg() {

      BYTE  idReg;
      char *      szStop;

      idReg = (BYTE) strtol(szRegister, &szStop, 10);
      idData = (BYTE) strtol(szByte, &szStop, 10);


      // DEPP API Call: DeppPutReg
      if(!DeppPutReg(hif, idReg, idData, fFalse)) {
            printf("DeppPutReg failed\n");
            return;
      }

      return;
}

/* ------------------------------------------------------------ */
/***  FParseParam
**
**    Parameters:
**          cszArg            - number of command line arguments
**          rgszArg           - array of command line argument strings
**
**    Return Value:
**          none
**
**    Errors:
**          Returns fTrue if not parse errors, fFalse if command line
**          errors detected.
**
**    Description:
**          Parse the command line parameters and set global variables
**          based on what is found.
*/

BOOL FParseParam(int cszArg, char * rgszArg[]) {

      int         iszArg;
```

```
	/* Initialize default flag values */
	fDvc		= fFalse;
	fFile		= fFalse;
	//fFile1		= fFalse;
	fByte		= fFalse;

	// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE

	fSendDataToAtlys  = fFalse;
	fGetDataFromAtlys = fFalse;
	fSend4BFPGA		= fFalse;
	fGet4BFPGA		= fFalse;
//	fCompareData		= fFalse;
//	fGenRandData		= fFalse;

	// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE

	// Ensure sufficient paramaters. Need at least program name and
action flag
	if (cszArg < 3) {
		return fFalse;
	}

	/* The first parameter is the action to perform. Copy the
	** the first parameter into the action string.
	*/
	StrcpyS(szAction, cchSzLen, rgszArg[1]);

	// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE

	if( strcmp(szAction, "-g") == 0) {
		fGet4BFPGA  = fTrue;
	}

	else if( strcmp(szAction, "-p") == 0) {
		fSend4BFPGA = fTrue;
	}

	/*else if( strcmp(szAction, "-r") == 0) {
		fGenRandData	= fTrue;
	}*/

	else if( strcmp(szAction, "-s") == 0) {
		fSendDataToAtlys  = fTrue;
	}

	else if( strcmp(szAction, "-r") == 0) {
		fGetDataFromAtlys = fTrue;
	}

	/*else if( strcmp(szAction, "-c") == 0) {
		fCompareData	= fTrue;
	}*/

	// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE

	else { // unrecognized action
		return fFalse;
	}
```

```
	/* Second paramater is target register on device. Copy second
	** paramater to the register string */
	StrcpyS(szRegister, cchSzLen, rgszArg[2]);


	/* Parse the command line parameters.
	*/
	iszArg = 3;
	while(iszArg < cszArg) {

		/* Check for the -d parameter which is used to specify
		** the device name of the device to query.
		*/
		if (strcmp(rgszArg[iszArg], "-d") == 0) {
			iszArg += 1;
			if (iszArg >= cszArg) {
				return fFalse;
			}
			StrcpyS(szDvc, cchUsrNameMax, rgszArg[iszArg++]);
			fDvc = fTrue;
		}

		/* Check for the -f parameter used to specify the
		** input/output file name.
		*/
		else if (strcmp(rgszArg[iszArg], "-f") == 0) {
			iszArg += 1;
			if (iszArg >= cszArg) {
				return fFalse;
			}
			StrcpyS(szFile,          cchSzLen          /*cchSnMax*/,
rgszArg[iszArg++]);
			fFile = fTrue;
		}

		/* Check for the -f1 parameter used to specify the
		** comparison file name.
		*/
		/*else if (strcmp(rgszArg[iszArg], "-f1") == 0) {
			iszArg += 1;
			if (iszArg >= cszArg) {
				return fFalse;
			}
			StrcpyS(szFile1, cchSnMax, rgszArg[iszArg++]);
			fFile1 = fTrue;
		}*/

		/* Check for the -b paramater used to specify the
		** value of a single data byte to be written to the register
		*/
		else if (strcmp(rgszArg[iszArg], "-b") == 0) {
			iszArg += 1;
			if (iszArg >= cszArg) {
				return fFalse;
			}
			StrcpyS(szByte,       cchSzLen       /*cchUsrNameMax*/,
rgszArg[iszArg++]);
			fByte = fTrue;
```

```
            }

            /* Not a recognized parameter
            */
            else {
                  return fFalse;
            }
      } // End while

      /* Input combination validity checks
      */
      if(!fDvc) {
            printf("Error: No device specified\n");
            return fFalse;
      }
      if( fSend4BFPGA && !fByte ) {
            printf("Error: No byte value provided\n");
            return fFalse;
      }

      if( fSendDataToAtlys && !fByte ) {
            printf("Error: No number of bytes to write provided\n");
            return fFalse;
      }

      if( fGetDataFromAtlys && !fByte ) {
            printf("Error: No number of bytes to read provided\n");
            return fFalse;
      }

      if(  fSendDataToAtlys && !fFile ) {
            printf("Error: No filename provided\n");
            return fFalse;
      }

      if(  fGetDataFromAtlys && !fFile ) {
            printf("Error: No filename provided\n");
            return fFalse;
      }

      /*if(  fCompareData && !fFile && !fFile1) {
            printf("Error: No filename provided\n");
            return fFalse;
      }*/

      return fTrue;

}

/* ------------------------------------------------------------ */
/***  ShowUsage
**
**    Synopsis
**          VOID ShowUsage(sz)
**
**    Input:
**          szProgName  - program name as called
**
**    Output:
```

```
**          none
**
**    Errors:
**          none
**
**    Description:
**          prints message to user detailing command line options
*/

void ShowUsage(char * szProgName) {

      printf("\nDigilent DEPP demo\n");
      printf("Usage: %s <action> <address> -d <device name> [options]\n",
szProgName);

      // NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE

      printf("\n\tActions:\n");
      printf("\t-p\tPut 32 bit value into Atlys's memories\n");
      printf("\t\tOptions: -b <value>\tValue to load into memory\n\n");
      printf("\t-g\tGet 32 bit value from Atlys's memories\n\n");
      //printf("\t-r\t\t\tGenerate 32 bit random value list\n");
      printf("\t-s\tSend Values to Atlys\n");
      printf("\t\tOptions:  -f  <filename>\tSpecify  file  name  to  read
(txt)\n");
      printf("\t\t\t -b <ammount>\tNumber or words to write\n\n");
      printf("\t-r\tGet Values from Atlys\n");
      printf("\t\tOptions:  -f  <filename>\tSpecify  file  name  to  write
(txt)\n");
      printf("\t\t\t -b <ammount>\tNumber or words to read\n\n");
      /*printf("\t-c\tCompare Values Sent With Those Received\n");
      printf("\t\tOptions: -f <filename>\tSpecify file name with values,
'*.txt'\n");
      printf("\t\t\t   -f1   <filename>\tSpecify   comparison   file   name,
'*.txt'\n");*/

      // END OF NEW CODE - END OF NEW CODE - END OF NEW CODE

      printf("\n\n");
}


/* ------------------------------------------------------------ */
/***  ErrorExit
**
**    Parameters:
**          none
**
**    Return Value:
**          none
**
**    Errors:
**          none
**
**    Description:
**          Disables DEPP, closes the device, close any open files, and
exits the program
*/
void ErrorExit() {
```

```
	if( hif != hifInvalid ) {
		// DEPP API Call: DeppDisable
		DeppDisable(hif);

		// DMGR API Call: DmgrClose
		DmgrClose(hif);
	}

	if( fhin != NULL ) {
		fclose(fhin);
	}

	if( fhout != NULL ) {
		fclose(fhout);
	}

	exit(1);
}

/* ------------------------------------------------------------ */
/***	StrcpyS
**
**	Parameters:
**		szDst - pointer to the destination string
**		cchDst - size of destination string
**		szSrc - pointer to zero terminated source string
**
**	Return Value:
**		none
**
**	Errors:
**		none
**
**	Description:
**		Cross platform version of Windows function strcpy_s.
*/
void StrcpyS( char* szDst, size_t cchDst, const char* szSrc ) {

	strcpy_s(szDst, cchDst, szSrc);

}

/* ------------------------------------------------------------ */

// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW
CODE - NEW CODE
// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW
CODE - NEW CODE
// NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW CODE - NEW
CODE - NEW CODE

 /*void GenRandData()
{
	int lowbound, upbound;
	int k,i;
	int curvalue;
	FILE *filein;

	// core
```

```
	if((filein = fopen("dataTOatlys.txt","w"))==NULL)
	{
		printf("Error opening input file!\n");
		printf("File with values should be named datain.txt .\n");
		goto lFail;
	}

	printf("Input file successfully opened.\n");
	printf("This function will generate random values in an interval of
yout choosing\n");
	printf("There are however upper and lower boundaries that must not
be exceeded\n");
	printf("Upper boundary must be 2^32-1 = %d or lower\n",
UPPERLIMIT);
	printf("Lower boundary begins at %d and upwards\n", LOWERLIMIT);

	printf("Enter your upper boundary:\n");
	scanf("%u",&upbound);
	if ((upbound>UPPERLIMIT)||(upbound<(LOWERLIMIT+1)))
	{
		printf("Value entered is out of bounds, it should be between
1 and 65535\n");
		goto lFail;
	}

	printf("Enter your lower boundary:\n");
	scanf("%d",&lowbound);
	if
((lowbound>upbound)||(lowbound<LOWERLIMIT)||(lowbound>(UPPERLIMIT-1)))
	{
		printf("Value entered is out of bounds, it should be between
%d and %d and not lower than %d\n", LOWERLIMIT, UPPERLIMIT-1, upbound);
		goto lFail;
	}

	srand((unsigned)time(NULL));

	for (i=0; i<n; i++)
	{
		curvalue		=		lowbound+(int)(double(upbound-
lowbound+1)*rand()/(RAND_MAX+1.0));
		data[i]=curvalue;
		if((k=(fprintf(filein,"%u\n",curvalue)))<1)
		{
			printf("Erroneous value entered, operation failed.\n");
			goto lFail;
		}
	}

	printf("List of random values created. Do please check file
datain.txt for your results.\n");

	lFail:
		if (filein != NULL)
		{
			fclose(filein);
		}
		return;
}*/
```

```
/*********************************************************************
/

 void Send4BFPGA()
{
      unsigned int memind=0x0, memindoriginal = 0x0, memind_t=0x0;
      unsigned int memval=0x0, memval_t=0x0;
      int destination;

      /* Send 32 bits to Atlys using DoPutReg() function */

      memindoriginal = atoi(szRegister);             //read from stdin
      memind = memindoriginal;
      destination = memoryops(&memind,WORDBYTES);               //choose
target memory in Atlys, alter address if needed
      memval = atoi(szByte);                                 //read    from
stdin
      memind_t = (FILTER0&memind)>>24;
      _itoa(memind_t, szRegister, 10);
      memval_t = (FILTER0&memval)>>24;
      _itoa(memval_t, szByte, 10);
      DoPutReg();                                                // First
Byte sent
      memind_t = (FILTER1&memind)>>16;
      _itoa(memind_t, szRegister, 10);
      memval_t = (FILTER1&memval)>>16;
      _itoa(memval_t, szByte, 10);
      DoPutReg();                                                //
Second Byte sent
      memind_t = (FILTER2&memind)>>8;
      _itoa(memind_t, szRegister, 10);
      memval_t = (FILTER2&memval)>>8;
      _itoa(memval_t, szByte, 10);
      DoPutReg();                                                // Third
Byte sent
      memind_t = FILTER3&memind;
      _itoa(memind_t, szRegister, 10);
      memval_t = FILTER3&memval;
      _itoa(memval_t, szByte, 10);
      DoPutReg();                                                //
Fourth Byte sent

      if (destination == 1) printf("Complete. DDR index %d set with value
%d / 0x%08X.\n", memindoriginal ,memval, memval);
      else {
            printf("Complete. QuadSPI Flash index %d set with value %d /
0x%08X.\n", memindoriginal ,memval, memval);
            printf("Remember that in order to actually WRITE to Flash you
must run operation a) in your FPGA, using the UART to do it.\n");
      }
 }

/*********************************************************************
/

void Get4BFPGA(){

      unsigned int memind=0x0, memindoriginal = 0x0, memind_t=0x0;
```

```
	unsigned int memval=0x0,memval_t=0x0;
	int destination;

	/* Receive 32 bits from Atlys using DoGetReg() function */

	memindoriginal = atoi(szRegister);		//read from stdin
	memind = memindoriginal;
	destination = memoryops(&memind,WORDBYTES);	//choose	target
memory in Atlys, alter address if needed
	memind_t = (FILTER0&memind)>>24;
	printf("First octet of address is 0x%02X\n",memind_t);
	_itoa(memind_t, szRegister, 10);
	DoGetReg();						// First
Byte of address word sent
	memind_t = (FILTER1&memind)>>16;
	printf("First octet of address is 0x%02X\n",memind_t);
	_itoa(memind_t, szRegister, 10);
	DoGetReg();						//
Second Byte of address word sent
	memind_t = (FILTER2&memind)>>8;
	printf("First octet of address is 0x%02X\n",memind_t);
	_itoa(memind_t, szRegister, 10);
	DoGetReg();						// Third
Byte of address word sent
	memind_t = FILTER3&memind;
	printf("First octet of address is 0x%02X\n",memind_t);
	_itoa(memind_t, szRegister, 10);
	DoGetReg();						//
Fourth Byte of address word sent
	printf("first octet is 0x%02X\n",idData);
	memval = (idData<<24)&FILTER0;			// First Byte
of data word received
	DoGetReg();						//
Second Byte of data word received
	printf("second octet is 0x%02X\n",idData);
	memval += (idData<<16)&FILTER1;
	DoGetReg();						// Third
Byte of data word received
	printf("third octet is 0x%02X\n",idData);
	memval += (idData<<8)&FILTER2;
	DoGetReg();						//
Fourth Byte of data word received
	printf("fourth octet is 0x%02X\n",idData);
	memval += idData&FILTER3;

	if (destination == 1) printf("Complete. DDR index %d reported value
%d / 0x%08X.\n", memindoriginal ,memval, memval);
	else {
		printf("Complete. QuadSPI Flash index %d reported value %d /
0x%08X.\n", memindoriginal ,memval, memval);
		printf("Remember that this value is actually the value
resident in the QuadSPI Flash backup in the DDR memory.\n");
	}

}

/*********************************************************************
/
```

```
void SendDataToAtlys() {

	FILE *filein;
	unsigned int regval, index, addressoriginal, address, count, i,
valuefromfile;
	int j, destination;
	char temp;

	if((filein = fopen(szFile,"r"))==NULL) {
		printf("Error opening input file!\n");
		goto lFail;
	}

	printf("Opened file was ' %s '\n", szFile);

	addressoriginal = atoi(szRegister);
	//read from stdin, address is the point at which data will start to
be written
	address  = addressoriginal;
	count = atoi(szByte);
	//read from stdin, count is the ammount of 32 bit words from the
dataTOatlys.txt file that will be written
	destination = memoryops(&address, WORDBYTES*count);	//choose
target memory in Atlys, alter address if needed

	if (destination == 1) printf("Sending your values to Atlys DDR
starting from address 0x%08X, an amount of %d words of the value
list.\n", addressoriginal, count);
	else printf("Sending your values to Atlys QuadSPI Flash starting
from address 0x%08X, an amount of %d words of the value list.\n",
addressoriginal, count);
	printf("Please hold...\n");

	i=0;
	j=0;
	while ((!feof(filein))&&(i<(WORDBYTES*count))){
		fscanf(filein, "%d\n", &valuefromfile);

		//printf("Address will be 0x%08X / %u\n", address+i,
address+i);
		//printf("Value read 0x%08X / %u\n", valuefromfile,
valuefromfile);

		index = (FILTER0&(address+i))>>24;
		//printf("Address will be 0x%02X / %u\n", index);
		_itoa(index, szRegister, 10);
		regval = (FILTER0&valuefromfile)>>24;
		//printf("Value 0x%02X / %u\n", regval, regval);
		_itoa(regval, szByte, 10);
		DoPutReg();
		// First Byte sent

		index = (FILTER1&(address+i))>>16;
		//printf("Address will be 0x%02X / %u\n", index);
		_itoa(index, szRegister, 10);
		regval = (FILTER1&valuefromfile)>>16;
		//printf("Value 0x%02X / %u\n", regval, regval);
		_itoa(regval, szByte, 10);
```

```
            DoPutReg();
            // Second Byte sent

            index = (FILTER2&(address+i))>>8;
            //printf("Address will be 0x%02X / %u\n", index);
            _itoa(index, szRegister, 10);
            regval = (FILTER2&valuefromfile)>>8;
            //printf("Value 0x%02X / %u\n", regval, regval);
            _itoa(regval, szByte, 10);
            DoPutReg();
            // Third Byte sent

            index = FILTER3&(address+i);
            //printf("Address will be 0x%02X / %u\n", index);
            _itoa(index, szRegister, 10);
            regval = FILTER3&valuefromfile;
            //printf("Value 0x%02X / %u\n", regval, regval);
            _itoa(regval, szByte, 10);
            DoPutReg();
            // Fourth Byte sent

            i+=WORDBYTES;
            j++;
            //printf("Enter anything: \n");
            //scanf("%c\n",&temp);
      }

      if (j!=count) {
            printf("End Of File: operation completed for only %d
values\n", j);
      }
      else printf("Values successfully sent to the Atlys board.\n");

lFail:
      if (filein != NULL) {
            fclose(filein);
      }

}

/*********************************************************************
/

void GetDataFromAtlys() {

      FILE *fileout;
      unsigned int regval, index, addressoriginal, address, count, i,
valuefromfpga = 0x00;
      int destination, ret;
      char temp;

      if((fileout = fopen(szFile,"w"))==NULL)
      {
            printf("Error opening input file!\n");
            goto lFail;
      }
```

```
      addressoriginal = atoi(szRegister);
      //read from stdin, address is the point at which data will start to
be read
      address  = addressoriginal;
      count = atoi(szByte);
      //read from stdin, count is the ammount of 32 bit words that will
be read to dataFROMatlys.txt
      destination = memoryops(&address, WORDBYTES*count);        //choose
target memory in Atlys, alter address if needed

      if (destination == 1) printf("Reading your values from Atlys DDR
starting from address 0x%08X, an amount of %d words will be written to
the value list.\n", addressoriginal, count);
      else printf("Reading your values from Atlys QuadSPI Flash starting
from address 0x%08X, an amount of %d words will be written to the value
list.\n", addressoriginal, count);
      printf("Please hold...\n");

      /* Receive data from FPGA */
      i=0;
      while (i<(WORDBYTES*count)){

            //printf("Address  will  be  0x%08X  /  %u\n",  address+i,
address+i);

            index = (FILTER0&(address+i))>>24;
            _itoa(index, szRegister, 10);
            DoGetReg();                                          // First
Byte of address word sent
            //printf("Address will be 0x%02X / %u\n", index, index);

            index = (FILTER1&(address+i))>>16;
            _itoa(index, szRegister, 10);
            DoGetReg();                                          //
Second Byte of address word sent
            //printf("Address will be 0x%02X / %u\n", index, index);

            index = (FILTER2&(address+i))>>8;
            _itoa(index, szRegister, 10);
            DoGetReg();                                          // Third
Byte of address word sent
            //printf("Address will be 0x%02X / %u\n", index, index);

            index = FILTER3&(address+i);
            _itoa(i, szRegister, 10);
            DoGetReg();                                          //
Fourth Byte of address word sent
            //printf("Address will be 0x%02X / %u\n", index, index);
            regval = (idData<<24)&FILTER0;                 //  First  Byte
of data word received
            //printf("Value 0x%08X / %u\n", regval, regval);

            DoGetReg();                                          //
Second Byte of data word received
            regval += (idData<<16)&FILTER1;
            //printf("Value 0x%08X / %u\n", regval, regval);

            DoGetReg();                                          // Third
Byte of data word received
```

```
            regval += (idData<<8)&FILTER2;
            //printf("Value 0x%08X / %u\n", regval, regval);

            DoGetReg();                                                    //
Fourth Byte of data word received
            valuefromfpga = regval + (idData&FILTER3);
            //printf("Value from fpga is 0x%08X / %u\n", valuefromfpga,
valuefromfpga);

            if((ret=(fprintf(fileout,"%u\n",valuefromfpga)))<1) {
                  printf("Erroneous value entered, operation failed.\n");
                  goto lFail;
            }

            i+=WORDBYTES;

            //printf("Enter anything: \n");
            //scanf("%c\n",&temp);
      }

      printf("Values successfully received and written to the output file
of your choosing.\n");

lFail:
      if (fileout != NULL) {
            fclose(fileout);
      }

}

/*********************************************************************
/
/*
 void CompareData()
{
      FILE *filein1;
      FILE *filein2;
      FILE *fileout;
      int i,k;
      int comptrue=0,compfalse=1;
      int errorcount=0;

      if((filein1 = fopen(szFile,"r"))==NULL)
      {
            printf("Error opening input file!\n");
            printf("File     with     values     should     be     named
dataTOatlys.txt.\n");
            goto lFail;
      }

      if((filein2 = fopen(szFile1,"r"))==NULL)
      {
            printf("Error opening input file!\n");
            printf("File     with     values     should     be     named
dataFROMatlys.txt.\n");
            goto lFail;
      }

      if((fileout = fopen("COMPARISON.txt","w"))==NULL)
```

```
	{
		printf("Error opening input file!\n");
		printf("File with values should be named COMPARISON.txt.\n");
		goto lFail;
	}


	for (i=0;i<=(n-1);i++)
	{
		fscanf(filein1,"%d\n",&data[i]);
		fscanf(filein2,"%d\n",&dataFPGA[i]);

		if (data[i] == dataFPGA[i])
		{
			if((k=(fprintf(fileout,"%d\n",comptrue)))<1)
			{
				printf("Erroneous	value	entered,	operation
failed.\n");
				goto lFail;
			}
		}
		else
		{
			errorcount++;
			if((k=(fprintf(fileout,"%d\n",compfalse)))<1)
			{
				printf("Erroneous	value	entered,	operation
failed.\n");
				goto lFail;
			}
		}

	}

	printf("Data	comparison	yielded	%d	errors	out	of	%d
values\n",errorcount,n);

	lFail:
	if ((filein1 != NULL) && (filein2 != NULL) && (fileout != NULL))
	{
		fclose(filein1);
		fclose(filein2);
		fclose(fileout);
	}

}*/

/*********************************************************************
/

int memoryops(UINT32 *address,int count) {
	char mem;

	printf("\r\nDo wish a to address the DDR or the QuadSPI Flash? (a
OR b)\r\n");
rep:printf("Enter your choice:\n");
	printf("\t:");
	scanf("%c",&mem);
	if (mem == 'A') mem = 'a';
```

```
      if (mem == 'B') mem = 'b';
      if ((mem!='a')&(mem!='b')) {
            printf("Value entered is not a valid option, please try
again.\n");
            goto rep;
      }
      switch (mem){
      case 'a':                                  //DDR addresses
            if (*address > CHANGEMAPBASEADDRESS) {
      //above DDR area addressable
                  printf("\nSelected address targets a reserved DDR-RAM
Memory area, Op will now exit.\n");
                  DeppDisable(hif);
                  DmgrClose(hif);
                  ErrorExit();
            }
            if ((*address + count) > CHANGEMAPBASEADDRESS) {     //above
DDR area addressable
                  printf("\nData partly targets a reserved DDR-RAM Memory
area, Op will now exit.\n");
                  DeppDisable(hif);
                  DmgrClose(hif);
                  ErrorExit();
            }
            break;
      case 'b':                                  //QuadSPI Flash
            if (*address > MAXFLASHADDRESS) {   //above QuadSPI Flash area
addressable
                  printf("\nSelected address targets above the QuadSPI
Flash size, Op will now exit.\n");
                  DeppDisable(hif);
                  DmgrClose(hif);
                  ErrorExit();
            }
            if ((*address + count) > MAXFLASHADDRESS) {    //above    DDR
area addressable
                  printf("\nData partly targets above the QuadSPI Flash
size, Op will now exit.\n");
                  DeppDisable(hif);
                  DmgrClose(hif);
                  ErrorExit();
            }
            *address += FLASHBACKUPBASEADDRESS;
            break;
      }
      if (mem == 'a') return 1;
      else return 0;
}


// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE - END OF NEW CODE
// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE - END OF NEW CODE
// END OF NEW CODE - END OF NEW CODE - END OF NEW CODE - END OF NEW CODE
```